DataGeneral

Business BASIC

Reference Manual

093-000137-03

Business BASIC

Reference Manual

093-000137-03

For the latest enhancements, cautions, documentation changes, and other information on this product, please see the Release Notice (085-series) supplied with the software.

Ordering No. 093-000137
© Data General Corporation, 1977, 1978
All Rights Reserved
Printed in the United States of America
Revision 03, September 1978
Licensed Material - Property of Data General Corporation

NOTICE

Data General Corporation (DGC) has prepared this manual for use by DGC personnel, licensees, and customers. The information contained herein is the property of DGC and shall not be reproduced in whole or in part without DGC prior written approval.

DGC reserves the right to make changes without notice in the specifications and materials contained herein and shall not be responsible for any damages (including consequential) caused by reliance on the materials presented, including but not limited to typographical, arithmetic, or listing errors.

Business BASIC
Reference Manual
093-000137

Revision History:

Original Release - February 1977
First Revision - September 1977
Second Revision - March 1978
Third Revision - September 1978

This document has been extensively revised from revision 02; therefore, change indicators have not been used.

The following are trademarks of Data General Corporation, Westboro, Massachusetts:

U.S. Registered Trademarks			Trademarks
CONTOUR I	INFOS	NOVALITE	DASHER
DATAPREP	NOVA	SUPERNOVA	DG/L
ECLIPSE	NOVADISC		microNOVA

Errata Sheet

Please replace the erroneous information in the text with the corrections as listed below.

Chapter/Page	Column	Paragraph	Correction
3-23	1	Why Use It	In line 2, change "market" to "marker."
3-23	1	How To Use It	In line 4, change "text" to "test."
3-31	1	What It Does	In line 2, change "the file pointer..." to "where the file pointer"
3-36	2	How To Use It - par 2	In line 7, change STRVAR to *strvar.*
3-47	2	What It Does - par 1	In line 4, change "the ame file" to "the same file".
3-65	2	Why Use It	In line 6, "likes" should be "lines".
3-73	2	What It Does	Line 14 should be "per array element for the range" so delete the second "element for the range".
3-88	1	par 2	Change "In" to "If" in line 1.
3-91	1	julian-date	In line 3 change "1068" to 1968."
3-103	2	How To Use It	In line 6, "cremmed" should be "crammed".
4-4	1	par 2	Change "source input jobs" to "source input files".
4-4	2	1st new para.	"'C' is the variable you use to reference the array..." This should be changed to: "'C1' is the variable you use to reference the array..."
4-25	1	par 1	Should be "its number of *key* fields is 2..."
4-27	1	CRAND	After the CRAND command we missed: -- CREATE create a sequential file.
5-1	2	par 2	"that are open" should be "that is open"
5-6	2	par 2	The formula is missing one parenthesis and should be: screen# = ((page#)-1)*(# of record types)+((format#)-1)
5-14	2	3	The last line should read "of TABBUILD."
7-4	1	Scratch Variables	The explanation of XO should read "new *next* available record."
7-6	2	1	Line 5 should say "you can type *text* or control commands..."
7-11	2	What It Does	For clarification: you can execute EDIT by typing: EDIT) from the BASIC CLI. You can't RUN "EDIT") from the CLI.
7-21	1	Substring Contents	For substring location 1,1, change "Channel number of file you want to be stored" to "Channel number of file you want to be *sorted.*"
7-25	1	Scratch Variables	The explanation of XO should read: "new *next* available record."

Preface

Data General's Business BASIC is a powerful, interactive language that allows many different users to run programs concurrently at a single computer installation. In a typical multiuser environment, you could be running a text editor while other users are running File Maintenance, Screen Maintenance, and other application programs. At the same time, the system could process data entry and retrieval operations. Business BASIC also lets you tailor a security system to meet your needs, with account and password control. Business BASIC provides application packages (utilities, subroutines, and CLI commands) which could alone solve many of your data processing problems; since they are written in BASIC you can also tailor these applications to suit your needs and improve your throughput. Included with Business BASIC is a simple ISAM file structure that maintains many index files which point to data files, and has the ability to handle subfiles; you can even use these features with a minimal hardware configuration. If you also have an ECLIPSE computer with RDOS INFOS® , Business BASIC provides an optional INFOS interface to handle INFOS file structures. Business BASIC presently runs on any DGC computer with an RDOS or DOS operating system.

This manual presumes that you have had some experience with the BASIC language. If you have not, read the introductory manual, *Basic BASIC,* for the background you need.

In a multiuser environment, a system often has a system manager, who is assisted by a system operator or operators. Most systems of this sort also distinguish between users such as a system programmer, an account manager, and a system user. This manual contains all the information you need to run a Business BASIC system, but you might want to separate the information intended specifically for system managers and system operators (Chapter 10) from the rest of the book which is a reference and tutorial manual for all users. If you manage a large system with many different types of users, you should reserve the privileged information in Chapter 10 for yourself.

Chapter 1 begins with an introduction to the system, and tells you how to log on and off, how to use interactive keyboard mode, how to write and edit program statements, and how to run programs.

Chapter 2 describes the Business BASIC components: statements, commands, subroutines, utilities, BASIC CLI commands, variables, expressions, arrays, and strings. It also describes Business BASIC arithmetic and provides a quick summary of key words and their suggested applications.

In Chapter 3, all interactive BASIC commands, statements, and functions, including file I/O statements, are listed alphabetically; each is explained in detail, and illustrated by an example. Before you use any of the sophisticated file I/O and ISAM statements, you should read through Chapter 4. INFOS interface statements are separately described in Chapter 9.

Chapter 4 contains all file information (except INFOS). It teaches you how to create files and subfiles in your programs, and it shows a variety of file access techniques, including index file access using Business BASIC's ISAM file structure.

Chapter 5 explains the File Maintenance (FM) application package. You should read this chapter (as well as Chapter 4) if you are planning file structures. Even if you are only going to use portions of the FM application package, you should read this chapter.

Chapter 6 explains the Screen Maintenance (SM) application package, and teaches you how to use formatted screen files for data entry/retrieval programs. It also shows you the format of the screen file maintained by SM; you can use this information to modify the package to suit your own needs.

Chapter 7 contains all the utility programs and subroutines provided with Business BASIC; they are listed in alphabetical order, explained in detail and illustrated by an example. You should first read Chapter 2 for a general description of subroutines and utilities. For information on how you use subroutines and utilities with file access (Chapter 4), File Maintenance (Chapter 5), or Screen Maintenance (Chapter 5), see the appropriate chapter.

Chapter 8 contains all the BASIC CLI commands in alphabetical order, described and illustrated with examples. The BASIC CLI is patterned after the RDOS/DOS CLI, so you should read the *RDOS/DOS CLI User's Manual* if you want to be more familiar with CLI operations.

Licensed Material-Property of Data General Corporation

Chapter 9 contains the Business BASIC INFOS interface statements and packet definitions. You *must* be familiar with and have an RDOS INFOS system on an ECLIPSE computer to use the INFOS interface. Read the *INFOS System User's Manual (RDOS)* for INFOS information.

Chapter 10 contains privileged information for the system manager and system operator. It includes Business BASIC system generation, the BSG dialog, system security, the system operator's CLI (OPCLI), privileged system calls, and automatic job execution. The privileged commands and other keywords are *not* included in the keyword summary which you will find in Appendix B. You should remove this chapter if you want to keep other users from reading it.

Appendix A is an ASCII character set chart. Appendix B is a keyword summary that lists each Business BASIC keyword in alphabetical order with its format, a simple explanation, and a chapter reference. Privileged keywords from Chapter 10 are *not* included. All STMA's, cursor positioning expressions, FM, SM, EDIT, and DOC keywords are also included in Appendix B. Appendix C lists all the error messages, including RDOS/DOS, utility, and INFOS error messages associated with Business BASIC. Appendix D explains how to call an assembly language subroutine using Business BASIC's UCALL statement.

Business BASIC runs under either RDOS or DOS. If you are the system manager and you want a detailed description of RDOS or DOS, see the following manuals:

- *Introduction to RDOS* (093-000083)
- *How To Load and Generate Your RDOS System* (093-000188) to bring up an RDOS system.
- *DOS Reference Manual* (093-000201) to bring up a DOS system.

Any user who needs to know other pertinent information about BASIC, BATCH programming, and/or CLI operations should read the appropriate manual:

- *Basic BASIC* (069-000003) to introduce the BASIC language.
- *BATCH User's Manual* (093-000087) to do BATCH programming.
- *RDOS/DOS CLI User's Manual* (093-000109) to understand uses of the CLI.

If you want to use the optional INFOS interface, you need an ECLIPSE computer, an RDOS INFOS system, and the following manual:

- *INFOS System User's Manual (RDOS)* (093-000114)

Reader, Please Note:

We use these conventions for command formats in this manual:

COMMAND required *[optional]* ...

Where	Means
COMMAND	You must enter the command (or its accepted abbreviation) as shown.
required	You must enter some argument (such as a filename). Sometimes, we use: $\{\text{required}_1 \atop \text{required}_2\}$ which means you must enter *one* of the arguments. Don't enter the braces; they only set off the choice.
[optional]	You have the option of entering some argument. Don't enter the brackets; they only set off what's optional.
...	You may repeat the preceding entry or entries. The explanation will tell you exactly what you may repeat.

Additionally, we use certain symbols in special ways:

Symbol	Means
)	Press the NEW LINE or RETURN key on your terminal's keyboard.
□	Be sure to put a space here. (We use this only when we must; normally, you can see where to put spaces.)

All numbers are decimal unless we indicate otherwise; e.g., 35_8.

Finally, in examples of system interactions, we use:

THIS TYPEFACE TO SHOW YOUR ENTRY)
THIS TYPEFACE FOR SYSTEM RESPONSES AND QUERIES

We have included some BASIC examples as line printer listings. These examples were created interactively, but the colon (:) comments were added using the EDIT utility described in Chapter 7. You should read about the EDIT utility to find out how to generate these colon comments for your listing and source files.

End of Preface

Licensed Material-Property of Data General Corporation

Contents

Chapter 1 - Introduction

Logging On · · ·

Chapter 3 - Statements, Commands, and Functions (continued)

CLOSE 3-14
CON 3-15
CRM$ 3-16
DATA 3-17
DEF 3-18
DELAY 3-19
DELETE 3-19
DIM 3-20
END 3-21
ENTER 3-22
EOF 3-23
ERASE 3-23
ERM$ 3-24
FILL$ 3-25
FOR/NEXT 3-26
GOSUB/RETURN 3-28
GOTO 3-30
GPOS 3-31
IF 3-32
INPUT 3-34
INPUT USING 3-36
INT 3-38
KADD 3-38
KDEL 3-40
KFIND 3-41
KNEXT 3-42
LEN 3-43
LET 3-44
LIST 3-45
LOAD 3-46
LOCK/UNLOCK 3-47
MAX 3-49
MIN 3-49
MOD 3-50
MSG 3-51
MTDIO 3-52
NEW 3-56
ON ERR 3-56
ON GOTO/ON GOSUB 3-57
ON IKEY 3-59
OPEN FILE 3-60
OR 3-62
PAGE 3-64
POSITION FILE 3-64
PRINT 3-65
PRINT USING 3-70
RANDOMIZE 3-72
READ 3-72
READ FILE 3-73
REM 3-75
RENAME 3-76
RENUMBER 3-76
REPLACE 3-78
RESTORE 3-79
RND 3-80

Licensed Material-Property of Data General Corporation

Chapter 3 - Statements, Commands, and Functions (continued)

RUN · · · · · 3-81
SAVE · · · · · 3-82
SGN · · · · · 3-83
SHFT · · · · · 3-84
SIZE · · · · · 3-85
SQR · · · · · 3-85
STMA · · · · · 3-86
STOP · · · · · 3-96
SWAP · · · · · 3-96
SYS · · · · · 3-98
TAB · · · · · 3-100
TINPUT · · · · · 3-101
TRUN$ · · · · · 3-102
UCM$ · · · · · 3-103
WRITE FILE · · · · · 3-104

Chapter 4 - Files

File Concepts · · · · · 4-1
- Filenames · · · · · 4-1
- File Attributes · · · · · 4-3
- Sequential Files · · · · · 4-3
- Random Files · · · · · 4-3
- Contiguous Files · · · · · 4-4
- Subfiles · · · · · 4-4
- Dynamic Record Allocation · · · · · 4-4
- Index Files · · · · · 4-6

File Setup · · · · · 4-9
- File Creation · · · · · 4-9
- PARAM file · · · · · 4-9
- INITFILE Utility · · · · · 4-11
- Example 4-1 · · · · · 4-11
- Example 4-2 · · · · · 4-12
- Example 4-3 · · · · · 4-12
- File Characteristics Array (C1) · · · · · 4-12
- Example 4-4 · · · · · 4-14
- OPEN Utility · · · · · 4-15
- Example 4-5 · · · · · 4-17
- Creating Index Files · · · · · 4-18
  - INITFILE · · · · · 4-18
- Example 4-6 · · · · · 4-19
- Example 4-7 · · · · · 4-20
  - INDEXBLD · · · · · 4-20
- Example 4-8 · · · · · 4-21
  - IBUILD · · · · · 4-22
- Example 4-9 · · · · · 4-24
  - TBUILD · · · · · 4-25
  - XBUILD · · · · · 4-25

Licensed Material-Property of Data General Corporation

Chapter 4 - Files (continued)

File Access 4-26
Input/Output Variables 4-26
File Access Keywords 4-26
Sequential Access 4-28
Example 4-10 4-28
Direct (Random) Access 4-29
Example 4-11 4-29
Linked-Available-Record Access 4-30
Example 4-12 4-31
Example 4-13 4-31
Example 4-14 4-32
Block Input/Output 4-33
Example 4-15 4-33
Example 4-16 4-34
ISAM (Index File Access) 4-34
Example 4-17 4-36
Example 4-18 4-37
Example 4-19 4-38
Example 4-20 4-38
File Locking 4-39

Chapter 5 - File Maintenance

Setting up a Table File 5-1
Type 1 Records 5-4
Type 2 Records 5-6
Type 3 Records 5-7
Type 4 Records 5-7
Type 5 Records 5-8
Type 6 Records 5-9
Using FM 5-9
Using Function Keys 5-10
FIND 5-10
FIND NEXT 5-10
CHANGE 5-10
ADD 5-10
DELETE 5-11
CLEAR ERROR 5-11
CHANGE FORMAT 5-11
Example 5-1 5-11
CHANGE PAGE 5-11
STOP 5-11
Roll-mode FM 5-11
Printing a Table File List 5-12
Example 5-2 5-13
Example 5-3 5-13
Printing an FM Data File 5-14
Printing a Log File 5-14
Copy Table File Records 5-14
Quick Definition of Arrays 5-14
Example 5-4 5-15
Example 5-5 5-15
Example 5-6 5-15
Example 5-7 5-16
Example 5-8 5-17
Example 5-9 5-17

Licensed Material-Property of Data General Corporation

Chapter 6 - Screen Handling

Defining Your Screen 6-1
Special Characters for Formatted Input/Output 6-4
Using Your Screen 6-7
Example 6-2 6-8
Sample Program 6-9
Example 6-3 6-10
Sample Output Routine 6-11
Screen File Layout 6-14

Chapter 7 - Subroutines and Utilities

ATTACH 7-2
CLI 7-3
DELREC.SL 7-4
DIV.SL 7-5
DOC 7-6
DOCTOC 7-11
EDIT 7-11
FILESORT 7-21
FINDFILE.SL 7-22
FORM.SL 7-23
GETCM.SL 7-24
GETREC.SL 7-25
IBUILD 7-26
INDEXBLD 7-28
INDEXCALC 7-29
INDEXPRT 7-30
INITFILE 7-31
INITINDEX.SL 7-32
LOCKS 7-33
LSPEED 7-33
MUL.SL 7-34
OPEN 7-35
PORTS 7-36
POSFL.SL 7-37
PROTFORM.SL 7-38
QFILESORT 7-39
RELINK 7-40
RNAM 7-40
SCHANS 7-41
TBUILD 7-42
TERM 7-43
UCHANS 7-43
UNFORM.SL 7-44
VAL.SL 7-46
VAR 7-47
XBUILD 7-47

Licensed Material-Property of Data General Corporation

Chapter 8 - BASIC CLI Commands

APPEND 8-2
ASG 8-3
BLDCOM 8-3
BUILD 8-4
CCONT 8-5
CDIR 8-5
CHAIN 8-6
CHATR 8-6
CHLAT 8-7
CPART 8-8
CRAND 8-9
CREATE 8-9
DELETE 8-10
DIR 8-11
DISK 8-11
DUMP 8-12
EQUIV 8-13
FDUMP 8-13
FILCOM 8-14
FILES 8-15
FLOAD 8-15
FPRINT 8-16
FREE 8-17
GDIR 8-17
GSDIR 8-18
GSYS 8-18
GTOD 8-19
INIT 8-19
LIBRARY 8-20
LINK 8-21
LIST 8-22
LOAD 8-24
LSTCOM 8-25
LSTMERGE 8-26
MDIR 8-27
MOVE 8-27
POP 8-28
PRINT 8-29
PRTCOM 8-29
QUIT 8-30
RELEASE 8-31
RENAME 8-31
SDIR 8-32
SLINE 8-32
SPDIS 8-33
SPEBL 8-33
SPKILL 8-34
START 8-34
STAT 8-35
TABLE 8-37
TCOPY 8-38
TFER 8-39
TPRINT 8-40
TYPE 8-41
UNLINK 8-41
VFU 8-42
XFER 8-45

Licensed Material-Property of Data General Corporation

Chapter 9 - INFOS Interface

General Format 9-1
INFOS Statement Functions 9-1
- Close an INFOS File (CLOSE) 9-2
- Define a Subindex (DFSI) 9-2
- Delete an INFOS Record (DELETE) 9-2
- Delete a Subindex (DELSI) 9-3
- Link to an Existing Subindex (LNKSI) 9-3
- Open an INFOS File (OPEN) 9-3
- Read a Record or a Partial Record (READ) 9-3
- Reinstate a Logically Deleted Record (REINS) 9-3
- Retrieve a Subindex Definition (RETDF) 9-3
- Retrieve the Highest Key in an Index (RETHK) 9-3
- Retrieve a Key (RETKY) 9-3
- Retrieve Record Status (RETST) 9-4
- Rewrite a Record (REWRITE) 9-4
- Write a Record (WRITE) 9-4

Packet Definitions 9-4
- INFOS Example 9-5

Chapter 10 - System Management

Installation -- RDOS 10-1
- Loading from Magnetic Tape 10-1
- Loading from Diskettes 10-1

Installation -- DOS 10-2
Generation and BSG 10-2
- Global Switches 10-2
- Sample BSG Dialogs 10-7
- Example 10-1 10-7
- Example 10-2 10-8

Executing Business BASIC 10-10
- Generating the Push Space 10-10

System Security 10-10
- HELLO and Accounts 10-10
- ACCOUNTING 10-11
- Example 10-3 10-12
- The Menu Concept 10-13
- U$ERS 10-13

Automatic Job Execution 10-14
NEWS 10-14
OPCLI 10-15
- ALL and FALL 10-15
- BYE 10-15
- CSLICE 10-15
- IKEY 10-15
- ISLICE 10-15
- MSG and FMSG 10-15
- SHOW 10-15
- Example 10-4 10-16
- SYSDEC 10-16
- PRI 10-16

Chapter 10 - System Management (continued)

Privileged System Calls 10-16
STMB 10-16
STMC 10-18
STMD 10-20
Example 10-5 10-21
Shutting Down Business BASIC 10-21
Example 10-6 10-22
User Status Table 10-23

Appendix A - ASCII Chart

Appendix B - Keyword Summary

Appendix C - Error Messages

Business BASIC Error Messages C-1
Utility Program Error Messages C-2
RDOS/DOS and I/O Error Messages C-2
INFOS Error Messages C-4
CLI Error Messages C-5

Appendix D - Calling an Assembly Language Subroutine from Business BASIC (UCALL)

Character String Storage and Definitions D-1
Linking the Assembly Language Subroutine D-2

Licensed Material-Property of Data General Corporation

Tables

Table	Caption	
2-1	Keyword Summary	2-9
3-1	Summary of Statements, Commands, and Functions	3-1
3-2	Access Modes	3-60
4-1	File Extensions	4-2
4-2	OPEN Modes	4-16
5-1	Record Types in a Table File	5-2
5-2	FM Function Keys	5-4
6-1	Special Characters	6-4
6-2	Screen Maintenance Commands	6-6
6-3	Screen File Layout	6-14
7-1	Command Summary for EDIT	7-18
7-2	IBUILD String Contents	7-26
10-1	ACCOUNTING File Format	10-13
10-2	User Status Table Values	10-23

Licensed Material-Property of Data General Corporation

Illustrations

Figure	Caption	
2-1	Single and Double Precision	2-4
2-2	Triple Precision	2-4
2-3	One and Two Dimensional Arrays	2-4
3-1	Checking Bit Location 6	3-5
3-2	The Powers of 2	3-6
3-3	Status Values	3-53
3-4	Logical OR of 2 Numbers	3-62
3-5	Tab Zones	3-66
3-6	The SHFT Function	3-84
3-7	HOME SWAPs to REMOTE	3-87
4-1	Disk Partitioning	4-3
4-2	Record 0	4-5
4-3	Data Record	4-5
4-4	Deleted Record Chain	4-5
4-5	Block 0 of an Index File	4-7
4-6	Index Block and Entries	4-7
4-7	The Structure of an Index File	4-8
4-8	File Information of PARAM File	4-10
4-9	C1 Array	4-13
4-10	IBUILD String	4-23
4-11	Tag File Key Entry	4-25
4-12	Index File Descriptor	4-34
5-1	File Maintenance Templates	5-3
5-2	Type 1 Record	5-5
5-3	Type 1 Record with Control Information	5-5
5-4	Type 2 Record	5-6
5-5	Type 3 Record	5-7
5-6	Type 4 Record	5-7
5-7	Type 5 Record	5-8
5-8	Type 6 Record	5-9
5-9	Finding a Record in CUST Based on a Partial Key	5-10
6-1	A Sample Screen in a Screen File	6-1
6-2	Screen Maintenance Template	6-3
6-3	Screen for the Sample Program in Example 6-3	6-5
6-4	Screen for Sample Output Routine	6-11
6-5	Sample Output Routine	6-12
7-1	Using Insert Mode	7-17
7-2	Using Comment Mode	7-17
10-1	Sample BSG Dialog	10-3
D-1	String Variable Storage	D-1
D-2	Argument Control Word	D-2
D-3	Example of an Assembly Language Subroutine	D-3

Licensed Material-Property of Data General Corporation

Chapter 1
Introduction

Data General's Business BASIC is a powerful interactive programming language. It has many features that enhance BASIC's inherent abilities to handle complex and diverse applications. Debugging is easy when you can interactively write and run programs. Business BASIC's utility and subroutine programs provide a quick screen handling and file maintenance package, ISAM file structure, INFOS® interface, and a CLI for all users in a multiuser system.

Business BASIC operates under the Real-time Disk Operating System (RDOS) and the Diskette Operating System (DOS) for virtually any Data General computer configuration. To use the optional INFOS interface, you must have an INFOS system on an ECLIPSE computer; however, Business BASIC can provide ISAM indexing for any Data General computer.

Data General's Business BASIC is an implementation of the BASIC language as developed at Dartmouth College, and includes the following features:

- Program and keyboard modes
- String manipulation
- Format control
- File maintenance and screen handling
- Sequential, random (direct), and ISAM file structures
- Utility programs and subroutines
- BASIC CLI
- System security
- INFOS interface

Logging On

To log on to a Business BASIC system, hit the ESCape key. A question and answer dialog will follow:

TERMINAL TYPE:

This question is asked if the system doesn't recognize the terminal you are using. If you are using a DG 6052/6053, this question is not asked. If you are using a DG 6012, your terminal type is "3". If you're using some other terminal type, ask your system manager for the proper answer to this question.

ACCOUNT:

You answer with your account name. Account names are five characters long: the first two characters indicate the type of account, where:

AA system manager
AB system programmer
OP system operator
xx user (your own characters)

The next three characters are usually your own initials. You can type all five characters in response to the ACCOUNT question:

ACCOUNT: AADGC)

or you can type the first two characters, and Business BASIC will ask for the next three as your user ID:

ACCOUNT: AA) *USER:* DGC)

The) symbol represents a carriage return. Always use the carriage return key to terminate lines that you type in keyboard mode. When you've finished typing your account name, the system will ask for a password:

PASSWORD:

Respond to this question by typing your password, followed by a carriage return. When you type your password, the system will not display what you're typing, so that no one can see your password.

When you've typed your password, the system may or may not ask for a directory specifier. If the system asks for a directory name, type the name of the directory in which you want to work. The system will not ask for a directory if one is already specified for your account.

This completes log on. The system will display some information, then display an asterisk (*) prompt at the left side of a terminal line. This prompt signifies that you are in keyboard mode. Business BASIC has three states of being: off, on (keyboard mode), and executing (program mode).

Licensed Material-Property of Data General Corporation

Keyboard Mode

When you have the asterisk (*) prompt, you are in keyboard mode. In this mode you can either type a command that the computer will execute immediately, or you can prepare a program to be executed later. A *program* is a series of instructions to the computer; the computer executes the instructions when you run the program. To learn how to write BASIC programs, you should read *Basic BASIC*. You will find commands, statements, and other Business BASIC components described in Chapter 2.

Working Storage

Whenever you create a program by typing statements with line numbers, you are actually typing characters into your *working storage* area. When you type a program statement, the system immediately interprets the statement and adds it to any other statements you've already typed into working storage. The system turns your ASCII characters (characters you type in) into machine-readable binary code when it executes the program. If you typed a statement whose syntax is incorrect, the system immediately tells you this. You then can either retype the statement with its line number, or edit the statement line with a few simple editing commands.

Example 1-1 shows a syntax error.

Example 1-1

```
*10 PJINT "HELLO" )
ERROR 2- STATEMENT OR COMMAND SYNTAX IS INVALID
*
```

To change the line, we'd retype line 10:

```
*10 PRINT "HELLO" )
*
```


Licensed Material-Property of Data General Corporation

The system returns with an asterisk (*) prompt, indicating that the statement was interpreted as being correct. Statement number 10 now exists in working storage. If you typed another statement number 10, it would replace the existing statement number 10.

You can RUN a program that is in working storage, and examine the contents of working storage with the LIST command. You can enter another program (with different line numbers) to merge it with the program in working storage, or ENTER another program (with the same line numbers) and replace the program in working storage. Save a program in working storage on disk with a SAVE command, and bring a saved program from disk into working storage with the LOAD command. You can clear your working storage with the NEW command. We describe these commands individually in Chapter 3 (as well as in *Basic BASIC*.

Writing a Program

First, clear your working storage area with a NEW command. Now there are no statements with line numbers in working storage, and you can choose any line number for your first statement. For most of our examples, we number our statements by increments of ten providing plenty of space to insert new statements between existing statements. You can assign line numbers from 1 to 9999. You can type any line number and statement out of sequence, as we do in our example, and the statements will line up automatically in ascending line number order:

```
*NEW )
*10 PRINT "HELLO" )
*5 LET X=2 )
*20 LET Y=X*10 )
*40 PRINT X,Y )
*30 PRINT "THE VALUES OF X AND Y ARE:" )

*LIST )

0005 LET X=2
0010 PRINT "HELLO"
0020 LET Y=X*10
0030 PRINT "THE VALUES OF X AND Y ARE:"
0040 PRINT X,Y

*RUN )
HELLO
THE VALUES OF X AND Y ARE:
2   20
*
```

Our program has run to completion, but it is still in working storage. Included in working storage with the program statements are the values of the variables X and Y. If you wanted to know what values they have after the program has run, you'd use the PRINT command:

```
*PRINT X,Y ) 2   20
*
```

When a program has completed its execution, it stays in working storage until a NEW, ENTER, or LOAD occurs or until you RUN another program. You can print the value of any variable if the program is still in working storage.

To save our program, we use the SAVE command and we make up a name for our program: SAMPLE.

```
*SAVE "SAMPLE" )
*NEW )
```

We then type NEW to clear our working storage so that we can type a new program. SAMPLE now exists in your directory as a program file; it can be LOADed, or simply RUN as follows:

```
*LOAD "SAMPLE" )
*RUN )
```

or

```
*RUN "SAMPLE" )
```

Editing a Program

When you type an incorrect line, you can change it any number of ways. If you type a character and you want to erase it, use the RUBOUT key (called DELETE on some terminals). This key will erase the character immediately to the left of the cursor on the screen. To erase a line, you can use the backslash (\) key at the end of the line to make the system ignore the line (on some terminals, the key is CTRL-X). You can also use the interrupt key (called IKEY, usually the ESCape or LINE FEED key) to ignore a line while you're still typing it. If you typed a line that did not cause a BASIC error but it is still wrong for your purposes, there are several things you can do:

1. Retype the line with its line number, effectively deleting the old line from working storage. Here's an example:

```
*LIST )

0010 PRINT "HELLO"

*10 PRINT "SAMPLE PROGRAM" )
*LIST )

0010 PRINT "SAMPLE PROGRAM"

*
```


Licensed Material-Property of Data General Corporation

2. Type the line number and a carriage return. This is an abbreviation of the ERASE command, and will delete the line specified by the line number. Here's an example:

```
*LIST )
0010 PRINT "SAMPLE PROGRAM"
0020 LET X=2
*10 )
*LIST )
0020 LET X=2
*
```

3. Use the keyboard editing commands. When a line causes an error to occur, the line is put in an edit buffer. When you LIST the contents of working storage, the last line LISTed will remain in an edit buffer. If you LIST only one line, it will remain in an edit buffer for you to edit using the following editing commands:

   .A string

   append (add to the end) string (string of characters) to the line in the edit buffer, but keep the line in the edit buffer (i.e., do not pass it back to working storage for it to be interpreted).

   .E/string$_1$ /string$_2$ *[/G]*

   change the first occurrence of string$_1$ in the edit buffer to string$_2$ (if you use the optional */G* terminator, it will change *all* occurrences of string$_1$ in the edit buffer to string$_2$). The edited line is not passed to working storage; it is kept in the edit buffer for further editing.

   .I string

   change the line in the edit buffer to string. The new line is not passed to working storage; it is kept in the edit buffer for further editing.

   .P

   display the contents of the edit buffer, but do not pass them to working storage.

   The period (.) command sends whatever is in the edit buffer to working storage for it to be interpreted.

   .C/string$_1$ /string$_2$ *[/G]*

   the same as the .E command, except that the contents of the edit buffer (the edited line) is immediately passed to working storage.

Licensed Material-Property of Data General Corporation

Example 1-2 shows a sample editing session.

Example 1-2

```
*100 PRINT THIS IS A PRINT STATEMENT" )
ERROR 2- STATEMENT OR COMMAND SYNTAX IS INVALID

(Because of the error, the statement is placed in the edit buffer.)

*.P )
100 PRINT THIS IS A STATEMENT"
*.E/THIS/"THIS )
100 PRINT "THIS IS A PRINT STATEMENT"
*.)

0100 PRINT "THIS IS A PRINT STATEMENT"

*LIST 290 )

0290 PRINT "THE LAST AMOUNT WAS ";AMOUNT
*.P )
0290 PRINT "THE LAST AMOUNT WAS";AMOUNT
*.C/AMOUNT/AMT )

0290 PRINT "THE LAST AMT WAS";AMOUNT

(We only changed "amount" in one place; let's try that again.)

*.P )
ERROR 73- EDIT BUFFER IS EMPTY

(Because we used .C, the edited statement was automatically sent back to working storage.)

*LIST 290 )
0290 PRINT "THE LAST AMT WAS";AMOUNT
*.E/OUN// )
0290 PRINT "THE LAST AMT WAS";AMT
*.C/AMT/AMOUNT )

0290 PRINT "THE LAST AMOUNT WAS";AMT

*

(Now our statement is correct, and .C passed it back to working storage.)
```


Licensed Material-Property of Data General Corporation

Running and Interrupting Programs

When you finish preparing your program in working storage, you can RUN the program by typing:

```
*RUN )
```

The program will start executing from the lowest numbered line and proceed to the next higher numbered line. The program will stop only if it encounters a STOP or an END statement in the program, if you hit the interrupt key to interrupt execution, or if an error occurs.

If an error occurs in your program, the line number of the statement where the error occurred will be displayed with an error message. You can then LIST the line and see what caused the error. You can also display values of variables using the PRINT command, or assign values to variables with the LET command.

When you've changed the incorrect line and you want to continue running the program from the line you changed, you can type:

```
*RUN 100 )
```

where 100 is the line you changed. If you type RUN without a line number, execution will start from the beginning of the program, all values for variables will be cleared, and all variables will be reinitialized. Therefore, if you assign a new value to a variable after an error or interrupt occurred, you would want to either RUN at a line number (so the value isn't cleared from the variable) or use the command CON to continue execution from the point where it stopped.

The CONtinue command will continue execution of the program at the *next* line after the line that caused the error (or the line that was interrupted). If you wanted to re-execute the line that caused the error, you would have to use the RUN command with a line number. Example 1-3 shows the use of both commands.

Example 1-3

```
*LIST )
0010 LET X=2
0020 LET Y=X*10
0030 LET Z=Y+A
0040 PRINT Z
*RUN )
ERROR 17 AT 0030--UNASSIGNED VARIABLE
*LIST 30 )
0030 LET Z=Y+A
*A=4 )
*CON )
0
*RUN )
ERROR 17 AT 0030--UNASSIGNED VARIABLE
*A=4 )
*RUN 30 )
24
*RUN )
ERROR 17 AT 0030--UNASSIGNED VARIABLE
*25 LET A=4 )
*RUN 25 )
24
*
```


Licensed Material-Property of Data General Corporation

In Example 1-3, we first ran the program and got an error message because the variable A in line 30 had not been assigned a value. We gave it a value (note that the LET command does not require the keyword LET), but when we used the CON command, we started executing at line 40 -- and Z only had a value of zero. We assigned a value to A again and ran the program at line 30. We then wrote a new line 25 to handle the bug, and we ran the program starting at line 25 (X and Y retained their values).

The two statements, ON ERR and ON IKEY, trap errors and interrupts in your program. When you use ON ERR to trap an error, the program doesn't stop on an error -- it does what the ON ERR statement tells it to do. ON IKEY works the same way: if an interrupt occurs, the program doesn't stop -- it does what the ON IKEY statement tells it to do. To use these statements, you must first read their descriptions in Chapter 3.

Executing a BASIC CLI Command

You can execute a BASIC CLI command (see Chapter 8 for the BASIC CLI commands) from keyboard mode by first typing an exclamation point (!) after your asterisk (*) prompt and before the command. The exclamation point (!) is the BASIC CLI prompt. Here is an example:

```
*!GTOD )
17:34:12   5/24/78
*
```

In this example we executed the CLI command GTOD, which displays the time and date. The effect is the same as if you used the SWAP command to execute CLI. The CLI automatically pops you back to BASIC's keyboard mode when it is finished; it does not disturb the working storage. It *does,* however, close any open files and clear the common area (see Chapter 4 for a discussion of files and the common area).

You can execute more than one CLI command on a line, if you separate the commands with semicolons as shown in Example 1-4.

You should not use this feature if you want to execute several BASIC CLI commands, because the swap for each command slows down response time. If you want to use the BASIC CLI, you should either RUN "CLI", or SWAP or CHAIN "CLI" and then execute the CLI commands.

Logging Off

Logging off is the easiest thing you can do. Just type

```
*BYE )
```

The system will print sign-off information, and your terminal will be inactive again, until someone hits the ESCape key to log on.

Example 1-4

```
*!DISK;GTOD;GSYS )


   SECTORS          USED          LEFT        CONTIGUOUS
   BYTES          115,432        71,976          66,696
                59,101,184    36,851,712      34,148,352
   17:34:56
                                5/24/78
   BBASIC
```

End of Chapter

Licensed Material-Property of Data General Corporation

Chapter 2
Business BASIC Components

Business BASIC has three major components: statements, commands, and functions. In addition, Business BASIC provides subroutines and utility programs written in BASIC that do special tasks. Finally, Business BASIC has its own Command Line Interpreter (CLI) patterned after the RDOS/DOS CLI so that every user in a Business BASIC system can have an operating CLI.

Business BASIC uses constants and variables to represent data, expressions to combine variables and constants, strings and arrays to hold data, and files to store data. We briefly describe all of these concepts in this chapter. If you've never used BASIC, you should read *Basic BASIC* for a complete tutorial on the use of general BASIC statements, commands, functions, variables, strings, arrays, and numeric expressions.

Statements

Every program is made up of statements. Each program statement has a line number, a BASIC keyword, and arguments. When you type a BASIC statement, you must give it a line number, or else it will be interpreted as a command. When BASIC executes a program, it executes the statement with the lowest line number and then proceeds to the next higher line number unless a statement directs the computer elsewhere, as the GOTO and GOSUB statements do. Line numbers must be integers in the range 1 to 9999. Here are two examples of program statements:

```
0010 LET A=1000
0020 PRINT A/2
```

We have a LET statement on line 10 that assigns the constant 1000 to the variable A. We have a PRINT statement on line 20 that prints the expression A/2 (A divided by 2). Line numbers do not have to be numbered by ones -- you should number them in larger increments in case you have to insert additional lines between them.

Some statements, like LET, PRINT, and CHAIN, can also be keyboard mode commands. You'll find all statements, commands, and functions described in Chapter 3.

Commands

Commands tell the system to do something immediately (whereas programs are instructions the system follows when they are executed). When you are in keyboard mode (i.e., when you have the BASIC asterisk (*) prompt), you can type in a command. Some commands save your program in memory (SAVE, REPLACE, or LIST), some retrieve your program from memory (LOAD, ENTER, RUN, CHAIN, SWAP, etc.), and some combinations of commands perform calculations (LET and PRINT). Here are some examples of commands:

```
*SAVE "PROG102" )
```

This command saves the program currently in working storage under the name -- PROG102. PROG102 is now a program file in the current directory.

```
*NEW )
```

This command clears working storage so that you can write a new program, or enter an existing one.

```
*LOAD "PROG5" )
```

This command brings the SAVEd program -- PROG5 -- into working storage.

```
*RUN )
```

This command executes the program currently in working storage.

```
*LET A=25 )
*PRINT SQR(A) ) 5
*; SQR(25) + A ) 30
*
```

These commands together perform calculations using the square root (SQR) function.

NOTE: The semicolon (;) is an abbreviation of the PRINT command.

Some program statements can also be commands. All statements, commands, and functions are described in Chapter 3.

Licensed Material-Property of Data General Corporation

Functions

Business BASIC has many arithmetic and string functions that are built in and ready for use. You can use arithmetic functions as numeric expressions wherever numeric expressions are allowed. Here are examples:

```
*LET X-SQR(25) )
*LET Y=AND(X,4) )
*;MAX(X,Y)-ABS(SHFT(Y)) )    /
*
```

The numeric functions in Business BASIC are:

ABS	absolute value
AND	logical binary AND of two numbers
ASC	extract a number from a string
EOF	detect an end of file
GPOS	current position of a file pointer
INT	truncate to an integer
LEN	current length of a string
MAX	the larger of two numbers
MIN	the smaller of two numbers
MOD	the remainder after dividing two numbers
OR	logical binary OR of two numbers
RND	random number generator
SGN	sign of a number
SHFT	logical bit shift of a number
SYS	returns system information

Each function is described separately in Chapter 3.

Business BASIC also has string functions -- see "Strings" further in this chapter.

Subroutines

You can write your own subroutines (using GOSUB and RETURN statements), but Business BASIC already has subroutines written in BASIC that will handle special tasks. Business BASIC subroutines exist as source files (character format) with filenames that have a .SL extension. These subroutines have RETURN statements; you must GOSUB to an entry point in the subroutine. Before you do this, you must use the ENTER command (or statement) to merge the subroutine with your program. All line numbers for Business BASIC subroutines fall between 7500 and 9999 (be careful if you use these line numbers in a program that also uses a prewritten subroutine). ENTER is described in Chapter 3; and all subroutines are described in Chapter 7.

Subroutines use specific variables. Sometimes it is required that you assign values to variables before using the subroutine, and allocate storage for certain string variables. Some subroutines return values in specific variables. These specific variable names are only specific to the subroutine, not to the BASIC system, so you can use these variable names in your programs (there are no reserved variable names). But, you must be careful when you use a variable name in a program that also uses any of the supplied subroutines. Variables associated with a subroutine are described with that subroutine, as well as entry points and line numbers. Here is a list of the Business BASIC subroutines; each is described separately in Chapter 7:

DELREC.SL	delete a record in a linked-available-record file (dynamic allocation).
DIV.SL	calculate extended percentage.
FINDFILE.SL	find a subfile and build the file characteristics array (C1).
FORM.SL	formatted screen field input/output.
GETCM.SL	read the CLI COM file (write your own CLI command).
GETREC.SL	get an available record in a linked-available-record file (dynamic allocation).
INITINDEX.SL	initialize an index file.
MUL.SL	calculate the percent of a number.
POSFL.SL	position to a record or a byte within a record of a file.
PROTFORM.SL	protect/unprotect a screen field.
UNFORM.SL	unformatted screen field input/output.
VAL.SL	convert a string to a number.

Utilities

Utility programs are programs already written in Business BASIC, already SAVEd, and ready for use. You can use the CHAIN or SWAP statements in a program to execute a utility program, or you can RUN a utility program from keyboard mode, or execute a utility program from the BASIC CLI.

The BASIC CLI itself is a utility program that you RUN, CHAIN to, or SWAP to. The BASIC CLI has its own CLI commands patterned after the RDOS/DOS CLI commands. To use these commands, you should have a working knowledge of the RDOS/DOS CLI (see the *RDOS/DOS CLI User's Manual)* and RDOS/DOS operating sytems. All BASIC CLI commands are described in Chapter 8.

Two other utilities, File Maintenance (FM) and Screen Maintenance (SM) are discussed separately, in Chapters 5 and 6 respectively, because they are very large utility programs and each has its own special functional keywords. FM provides a fast access to any data file (with up to three index files) that was set up using FM. SM maintains screens and provides screen handling functions for specialized data entry and retrieval operations.

Licensed Material-Property of Data General Corporation

All utility programs except FM and SM are described in Chapter 7. Here's a list of utility programs:

ATTACH	attach yourself to a de-attached job.
CLI	put yourself in the BASIC CLI.
DOC	print a series of text files.
DOCTOC	print a table of contents for files used with DOC.
EDIT	create and/or edit text files.
FILESORT	sort a data file.
IBUILD	build an index file from a tag or data file.
INDEXBLD	build an index file.
INDEXCALC	calculate index file information.
INDEXPRT	print an index file.
INITFILE	initialize an index or data file.
LOCKS	display your current file locks.
LSPEED	change your terminal's speed.
OPEN	open files in your program (especially subfiles).
PORTS	display jobs on your system.
QFILESORT	quick sort of a data file.
RELINK	clean up a linked-available-record file.
RNAM	rename program variables.
SCHANS	display the system channel assignments.
TBUILD	build a tag file.
TERM	set certain terminal keys.
UCHANS	display your channel assignments.
VAR	list variables in a program.
XBUILD	build an index file from a data file.

Variables

Variables represent values. You can assign values to variables using READ and DATA statements, the LET statement, and the INPUT statement. You can change the value of a variable using these statements, and perform calculations with variables using other statements. File access statements (READ FILE, WRITE FILE, etc.) use variables to transfer data to and from files. Some supplied subroutines and utilities require you to assign values to specific variables before using them.

There are three types of variables: numeric, array, and string. Arrays are discussed in the next section, and string variables are discussed in "Strings".

Variable names must begin with a letter, and they can have up to 5 alphanumeric characters following the first letter, but only the last character can be a special character (e.g.,$, #, %). Here are some examples of legal and illegal variable names:

Legal	Illegal
A	1
A303	1AB
Z6B8	12345678
WAGE	2WAGE
WAGE2	$WAGE
WAGE$	#WAGE
WAGE%	%WAGE
WAGE#	A$%#

The last three legal variables (WAGE$, WAGE%, and WAGE#) represent special variable types. A dollar sign ($) at the end of a variable indicates that it is a string variable (see "Strings"). A percent sign (%) at the end indicates that it is a single precision numeric variable. A number sign (#) at the end indicates that it is a triple precision numeric variable. No sign indicates that the variable is a regular double precision numeric variable.

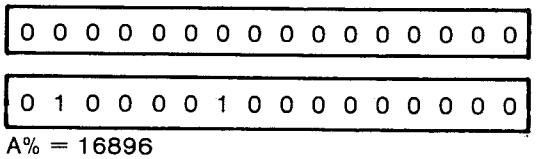
All numbers in Business BASIC must be integers. You cannot assign a fractional number to a variable. You can, however, use formatting techniques (PRINT USING, VAL.SL, etc.) to print numbers with decimal points. Single precision and double precision numeric variables store numbers as binary values using 32 bits (4 bytes, or 2 16-bit words). A bit is either *off* (0) or *on* (1), and a collection of bits form a binary (base 2) number that converts easily to an octal (base 8) number or a decimal (base 10) number. From now on, whenever we use the word *number* we will mean a decimal (base 10) integer unless we specify something else.

Figure 2-1 shows the binary representation (in bits) of a single and a double precision positive number. A single precision positive number only occupies one word: the lower (right-hand) 16 bits of a 32-bit number. Single precision variables need both words (32 bits), but a single precision variable will transfer only 16 bits from a program to a file and vice-versa, using READ FILE, WRITE FILE, BLOCK READ FILE, BLOCK WRITE FILE, INPUT FILE, and PRINT FILE. Business BASIC stores a negative number as the two's complement of its absolute value (for all precisions).

A single precision variable (%) should only contain numbers in the range -32767 to 32768; you can store a larger number in such a variable, but you cannot transfer that value to or from a file using that (%) variable.

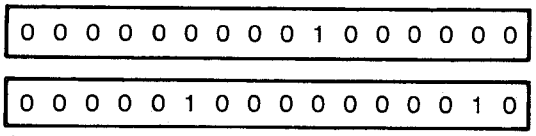
Figure 2-2 shows the binary representation of a triple precision number. You will only be using triple precision variables if your system is generated to allow them (see Chapter 10 for system manager information).

SINGLE PRECISION A%:



A% = 16896

DOUBLE PRECISION A:

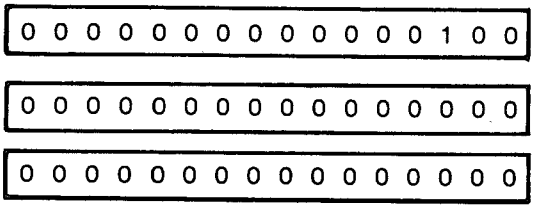


A = 4194304

SD-01172

Figure 2-1. Single and Double Precision

TRIPLE PRECISION A #:



A# = 17, 179, 869, 184(or 2↑34)

SD-01173

Figure 2-2. Triple Precision

```
*DIM A(5)
```

0	1	2	3	4	5

```
*LET A(0) = 0
*LET A(1) = 1
*LET A(2) = 2
*LET A(3) = 3
*LET A(4) = 4
*LET A(5) = 5

*DIM AB(3,2)
```

0	1	2
3	4	5
6	7	8
9	10	11

```
*LET AB(0,0) = 0
*LET AB(0,1) = 1
*LET AB(0,2) = 2
*LET AB(1,0) = 3
*LET AB(1,1) = 4
*LET AB(1,2) = 5
*LET AB(2,0) = 6
*LET AB(2,1) = 7
*LET AB(2,2) = 8
*LET AB(3,0) = 9
*LET AB(3,1) = 10
*LET AB(3,2) = 11
```

SD-01174

Figure 2-3. One and Two Dimensional Arrays

Arrays

An array is an ordered set of values. Each member of the set is called an array element. Each array element stores a 4-byte value (double precision numeric); if you are using triple precision variables, each array element will hold 6-byte values. Arrays are used like variables -- to represent values -- but a single array can hold as many values as you want. There is no restriction on the number of elements you can place in an array other than restrictions due to available memory.

Arrays can have one dimension, where each element forms a column of the array. Arrays can also have two dimensions, with rows and columns. Figure 2-3 shows both.

In Figure 2-3, we use a DIMension command to allocate space for the array. You'd use a DIMension statement in your program. One-dimensional arrays are always referenced by one subscript to the array variable, for example, A(1) refers to the second element. Two-dimensional arrays are referenced by two subscripts to the array variable -- if you only use one subscript, it specifies the 0 column of the row in the subscript. For example, AB(1,2) refers to row 1, column 2, but AB(1) refers to row 1, column 0. You can use parentheses or square brackets to enclose your subscripts. For two-dimensional arrays, the first subscript refers to the row, and the second subscript refers to the column. Notice that arrays start at 0: when you DIM A(5), you get a one-dimensional array that can hold 6 elements; when you DIM AB(3,2), you get a 4 by 3 two-dimensional array that can hold 12 elements.

Licensed Material-Property of Data General Corporation

You refer to an element in an array by specifying the row and column in a two-dimensional array, or the column in a one-dimensional array (e.g., in Figure 2-3, if X=A(4), X=4; if Y=AB(2,1), Y=7). Array variables follow the same rules as numeric variables, except that you can't have an array element that holds only 16 bits for a single precision (%) value; the array element will always hold at least 32 bits. For more information about arrays, see the DIM and LET statements.

You can redimension an array to its current size or smaller. For example, the two-dimensional array AB in Figure 2-3 could be redimensioned to DIM AB(2,3), because it would still hold 12 elements. Redimensioning an array does not change the values in the array, it just changes the location in the array of a certain value. If you redimension an array to have fewer elements, you will not free up the unused locations, you'll just make it impossible to refer to the unused locations.

Arithmetic

Numbers are stored as 32-bit integers, and can range from -2,147,483,647 to +2,147,483,647. If you're using triple precision numbers, they can range from +2 ↑ 46 to -(2 ↑ 46-1). Single precision variables (%) used to transfer 16 bits from file to program or program to file can only hold numbers in the range -32767 to +32768.

Expressions

A numeric expression is any combination of numbers, numeric variables, array elements, and numeric functions, linked together by arithmetic operators and parentheses. The arithmetic operators are:

Operator	Meaning	Example
+	Unary plus sign	A+(+B)
-	Unary minus sign	A+(-B)
↑	Exponentiation	A↑B (A to the power B)
*	Multiplication	A*B
/	Division	A/B
+	Addition	A+B+C
-	Subtraction	A-B

Business BASIC calculates numeric expressions in this order:

1. Any expressions in parentheses; the innermost expression in nested parentheses.
2. Unary plus and minus.
3. Exponents; left to right in a series of exponents.
4. Multiplication and division, left to right.
5. Addition and subtraction, left to right.

For example:

```
LET X=Z+(-A)+B*C↑D
```

The computer calculates X in this order:

1. A is negated.
2. C ↑ D is calculated.
3. B is multiplied by the result of step 2.
4. Z is added to the result of step 1.
5. The result of step 4 is added to the result of step 3.

Remember, triple precision variables (#) are allowed only if your system is generated with triple precision. Some legal and illegal triple precision assignments are:

Legal	Illegal
A#=B + C	A=B# + C
A#=B#+C#	
A#=B#+C	

Relational Operators

You use relational operators to compare two expressions in an IF decision statement. Relational operators are:

Operator	Meaning	Example
=	Equals	A=B
<	Less than	A<B
<=	Less than or equal to	A<=B
>	Greater than	A>B
>=	Greater than or equal to	A>=B
<>	Not equal	A<>B

Here are some examples:

```
0010 IF A ↑ B >= 32767 THEN GOTO 0300
0020 IF ABS(X*Y) <> A THEN GOSUB 0900
```

Strings

A string is a combination of characters. It may include letters, digits, spaces, binary values, and special characters. A string literal is a string enclosed within quotation marks; i.e., a string constant. You assign a string literal to a string variable as you would assign a numeric constant to a numeric variable. You can also use string literals in PRINT and INPUT statements. INPUT statements use the string literal as a prompt for the input query (see INPUT, Chapter 3). PRINT statements will simply print the string literal:

```
*PRINT "THIS IS A STRING" ) THIS IS A STRING
```

Here's an example of a string assignment:

```
0010 LET A$="THIS IS A STRING < 7 > "
```

The number "7" in angle brackets is the decimal ASCII character code for the terminal bell. When this statement is executed, the string *"THIS IS A STRING"* will appear on the terminal and you will hear a bell or beep. Every character has a decimal ASCII code number (see Appendix A), and you can use the decimal code number with angle brackets to put special and nonprinting ASCII characters into a string.

String Variables

String variables represent strings, and their names always have a dollar sign ($) suffix. String variables hold each character value in one byte (8 bits). Each character has an ASCII character code number that is stored in binary (base 2) in the 8 bits. You use the DIMension statement to allocate the maximum number of bytes (characters) for the string variable before you assign a string value to it.

String variables start at byte 1. Therefore,

```
0010 DIM A$(25),B3$(215)
```

dimensions two string variables: A$ has a maximum of 25 bytes, and B3$ has a maximum of 215 bytes. You can't assign to a string variable more characters (bytes) than its maximum length. If you redimension a string, your new maximum length must be less than or equal to the original maximum length.

Here are examples of string assignments:

```
0010 DIM A$(25),B3$(215)
0020 LET A$="THIS STRING HAS 24 CHAR."
0030 LET B3$(1,5)="12345"
```

Line 20 above almost fills A$ to its maximum length. A$'s current length, however, is 24. Line 30 fills part of B3$ -- from byte 1 to byte 5 for a total of 5 characters (each digit is a character). B3$'s current length is only 5. Line 30 is a *substring* assignment.

Substrings

You can select portions of a string by subscripting your string variable. If you specify one subscript, you are referring to the portion of the string from that byte position to the last byte position. If you specify two subscripts, the first will be the starting byte position, and the second the ending byte position of the substring. The subscript "0" refers to the byte position after the current length (e.g., for adding a substring to the end of a string).

For example:

A$	refers to the entire string.
A$(2)	refers to the 2nd through the last character in the string.
A$(I)	refers to character position I through the last character.
A$(3,7)	refers to the 3rd through the 7th character in the string.
A$(I,J)	refers to positions I through J, where I < = J.
A$(2,2)	refers to only the 2nd character in the string.
A$(0)	refers to the end of the string's current length, equivalent to A$(LEN(A$)+1).

Using subscripted variables, you can refer to substrings in LET, PRINT, INPUT, READ, and other statements. You can include string variables, subscripted string variables, string functions, and the concatenation operator (comma) in a string expression for an assignment (LET, READ) statement. You can use string variables, subscripted string variables, and the concatenation operator in PRINT and INPUT statements; but do *not* use string functions in PRINT and INPUT statements, only in assignment and comparison statements. See the following string assignments:

LET A$=B$	Replaces the contents of A$ with the contents of B$.
LET A$=""	Replaces the contents of A$ with a null string.
LET A$=A$,B$	Appends contents of B$ to the current contents of A$.
LET A$(0)=B$	Appends contents of B$ to the current contents of A$.
LET A$=B$,A$	Produces unpredictable results, since A$ becomes B$ before it can be appended to B$.
LET A$=4+5	Replaces contents of A$ with the digit 9.
LET A$=12345	Replaces contents of A$ with the string of digits "12345", *not* the number 12345.
LET A$=CHR$(12,4)	Replaces contents of A$ with a 4-byte string holding the binary value of the number 12.

The last assignment uses the CHR$ function to create a 4-byte string holding the binary value of a number. Remember, a string holds the binary ASCII value of a character in a byte, so a string of digits would be a string of the ASCII values. When you use the CHR$ function, you specify the number of bytes you are assigning, and you fill those bytes with the actual binary value of a number *not* with ASCII values. You would then extract that number from the string using the ASC function.

Licensed Material-Property of Data General Corporation

When you assign characters to a string of substring, the number of characters (bytes) assigned determines the current length. See Example 2-1.

When you use strings in relational expressions for IF statements, the strings are compared character by character on the basis of their ASCII values. If a character in a given position in one string has a higher ASCII value than the character in that same position in the other string, the first string is greater. If the characters in the same positions are identical but one string has more characters than the other, the longer string is greater. See Example 2-2.

Example 2-1

```
0010 DIM A$[512],B$[512]      :The maximum is 512 for both.
0020 LET B$=1                 :B$ holds the digit 1.
0030 LET A$="ABCDEF"          :Current length of A$ is 6.
0040 LET A$[3,3]=B$
0045 PRINT A$                 :Prints "AB1DEF".
0050 LET A$[3,6]=B$
0055 PRINT A$                 :Prints "AB1", current length is 3.
0060 LET A$[3]=B$
0065 PRINT A$                 :Prints "AB1"
0070 LET A$[3]=B$,B$,B$
0075 PRINT A$                 :Prints "AB111".
0080 PRINT LEN(A$)            :Prints 5.
```

Example 2-2

```
0010 DIM A$[512],B$[512]      :Maximum for both is 512.
0020 LET A$="ABCD"
0030 LET B$="BCDE"
0040 IF A$<B$ THEN PRINT "YES"          :Yes, A$ is less than B$.
0050 LET A$[1,1]=B$[4,4]                :Produces "EBCD" in A$.
0060 IF A$<B$ THEN PRINT "YES"          :No, because "E">"B".
0070 LET A$=B$,B$                       :Produces "BCDEBCDE" in A$.
0080 IF A$>B$ THEN PRINT "YES, A$ IS GREATER"    :Yes, A$>B$.
```


Licensed Material-Property of Data General Corporation

Business BASIC has several string functions that return string values, and two numeric functions that return numeric values about strings. The string functions are:

CHR$	Put the binary value of a number into a string.
CRM$	Cram every 3 bytes of string into 2 bytes.
ERM$	The string with the error message for the error code.
FILL$	Fill a string for a specified number of bytes with the character you specify.
TRUN$	Truncate a string.
UCM$	Uncram a crammed string (opposite of CRM$).

The numeric functions are:

ASC	Extract a number from a string (opposite of CHR$).
LEN	Determine the current length of a string.

Each function is discussed separately in Chapter 3.

Files

A file is a collection of data. You create a file whenever you store a program -- the program you save with a SAVE command and a filename is called a program file. If you LIST a program to a file, you are putting the ASCII characters into a character format file that can be printed or edited using the EDIT utility. We call these files text files, character format files, source files, or listing files -- a text or character format file contains text, a source file is a text file with a program in it (usually commented), and a listing file is a temporary source file (when using the EDIT utility to make comments).

You also have data files, index files, tag files, etc. These files usually hold binary data, and you read from or write to these files (called I/O or input/output). These files are described separately in Chapter 4.

Keywords

For a list of keywords in alphabetical order and a summary of their meanings, see Appendix B. The following chart lists keywords and their suggested applications.

We use these abbreviations:

S	Statement
C	Command
F	Function
Sub	Subroutine
Util	Utility
CLI	CLI Command

Licensed Material-Property of Data General Corporation

Table 2-1. Keyword Summary

Program Development								
Keyword	**Application**	**S**	**C**	**F**	**Sub**	**Util**	**CLI**	**Chapter**
CHAIN	execute a utility or another program from your program.	●	●				●	3,8
DATA	data for variables in READ.	●						3
DEF	define a user function.	●						3
DELAY	delay execution of the next command or statement.	●	●					3
DIM	dimension arrays and strings.	●	●					3
END	optional program terminator.	●						3
ENTER	bring in/merge source statements.	●	●					3
ERASE	erase lines in working storage.	●	●					3
FOR/NEXT	define a program loop.	●						3
GOSUB/RETURN	execute a subroutine.	●						3
GOTO	go to a specific statement.	●						3
IF	conditional transfer or execution.	●						3
LET	assign a value to a variable.	●	●					3
LIST	display contents of working storage.		●					3
LOAD	bring in a SAVEd program.		●					3
NEW	clear working storage.	●	●					3
ON ERR	trap an error in your program.	●						3
ON GOTO ON GOSUB	conditional transfer to a statement or subroutine.	● ●						3
ON IKEY	trap an interrupt in your program.	●						3
READ	assign DATA values to variables.	●						3
REM	remark statement.	●						3
RENUMBER	renumber program statements.		●					3
REPLACE	replace a SAVEd program with another.	●	●					3
RESTORE	reset the DATA list pointer.	●	●					3
RNAM	rename program variables.					●		7
RUN	execute a program.		●					3
SAVE	save a program in a program file.	●	●					3

Licensed Material-Property of Data General Corporation

Table 2-1. Keyword Summary (continued)

Program Development (continued)								
Keyword	**Application**	**S**	**C**	**F**	**Sub**	**Util**	**CLI**	**Chapter**
SIZE	determine size of program.		•					3
STMA	user system calls.	•	•					3
STOP	stop program execution.	•						3
SWAP	execute another program and return.	•	•					3
TABLE	print a program cross reference.						•	8
VAR	list variables.					•		7
File I/O								
Keyword	**Application**	**S**	**C**	**F**	**Sub**	**Util**	**CLI**	**Chapter**
BLOCK READ *[FILE]*	block input.	•	•					3
BLOCK WRITE *[FILE]*	block output.	•	•					3
CLOSE *[FILE]*	close any or all files.	•	•					3
DELREC.SL	delete a record.				•			7
EOF	end-of-file test.			•				3
GPOS	file position test.			•				3
GETREC.SL	get available record.				•			7
FORM.SL	formatted screen I/O.				•			7
INPUT *[FILE]*	input from file.	•	•					3
INPUT USING	input with error trap.	•	•					3
KADD	add a key to an index.	•	•					3
KDEL	delete a key from an index.	•	•					3
KFIND	find a key in an index.	•	•					3
KNEXT	find next key in index.	•	•					3
MTDIO	mag tape I/O.	•	•					3
OPEN	open several files.					•		7
OPEN FILE	open a file.	•	•					3
POSFL.SL	position to a record.				•			7
POSITION FILE	position to a byte.	•	•					3
PRINT *[FILE]*	output characters.	•	•					3
PRINT USING	output using a format.	•	•					3

Licensed Material-Property of Data General Corporation

S	Statement
C	Command
F	Function
Sub	Subroutine
Util	Utility
CLI	CLI Command

Table 2-1. Keyword Summary (continued)

File I/O (continued)								
Keyword	**Application**	**S**	**C**	**F**	**Sub**	**Util**	**CLI**	**Chapter**
PROTFORM.SL	protect/unprotect screen fields.				•			7
READ FILE	input from file.	•	•					3
TINPUT *[FILE]*	timed input.	•	•					3
WRITE FILE	output to file.	•	•					3
UNFORM.SL	unformatted I/O with screen files.				•			7
Maintain Files								
Keyword	**Application**	**S**	**C**	**F**	**Sub**	**Util**	**CLI**	**Chapter**
APPEND	append one file to another.						•	8
CCONT	create a contiguous file.						•	8
CRAND	create a random file.						•	8
CREATE	create a sequential file.						•	8
CHATR	change file attributes.						•	8
CHLAT	change link attributes.						•	8
DELETE	delete a file.	•	•				•	3,8
DOC & DOCTOC	documentation package.					•		7
DUMP	dump files.						•	8
EDIT	create/edit text files.					•		7
FILCOM	compare two files.						•	8
FILES	display files in directory.						•	8
FILESORT	sort a data file.					•		7
FDUMP	fast dump.						•	8
FLOAD	fast load.						•	8
FINDFILE.SL	find a subfile.				•			7

Licensed Material-Property of Data General Corporation

Table 2-1. Keyword Summary (continued)

Maintain Files (continued)								
Keyword	**Application**	**S**	**C**	**F**	**Sub**	**Util**	**CLI**	**Chapter**
FM	File Maintenance					•		5
FMPRINT	print a file.					•		5
FMTABPRINT	print a table file.					•		5
FMLOG	print a log file.					•		5
MOVETABREC	copy table records.					•		5
TABBUILD	define arrays.					•		5
FPRINT	print a file.						•	8
IBUILD	build an index file.					•		7
INDEXBLD	build or rebuild an index.					•		7
INDEXPRT	print an index file.					•		7
INDEXCALC	calculate index information.					•		7
INITFILE	initialize a file.					•		7
INITINDEX.SL	initialize an index.				•			7
LIBRARY	display filenames.						•	8
LINK	establish a link.						•	8
LIST	display file information.						•	8
LIST	output characters to file.		•					3
LOAD	load dumped files.						•	8
LOCK/UNLOCK	lock and unlock file blocks.	•	•					3
LOCKS	display current locks.					•		7
LSTCOM	compare listing files.						•	8
LSTMERGE	merge listing files.						•	8
MOVE	move file to another directory.						•	8
PRINT	print a text file.						•	8
QFILESORT	quick sort of a data file.					•		7
RELINK	relink a linked-available-record file.					•		7
RENAME	rename a file.		•				•	3,8
SCHANS	display system channels.					•		7
SM	maintain a screen file.					•		6
TBUILD	build a tag file.					•		7
TFER	copy a file to tape.						•	8

Licensed Material-Property of Data General Corporation

S	Statement
C	Command
F	Function
Sub	Subroutine
Util	Utility
CLI	CLI Command

Table 2-1. Keyword Summary (continued)

Maintain Files (continued)								
Keyword	**Application**	**S**	**C**	**F**	**Sub**	**Util**	**CLI**	**Chapter**
TPRINT	print a tuning report.						•	8
TYPE	display a text file.						•	8
UCHANS	display user channels.					•		7
UNLINK	delete a link reference.						•	8
XBUILD	build an index file.					•		7
XFER	copy file to another file.						•	8
VFU	create/edit a VFU file.						•	8
TCOPY	mag tape file copy.						•	8
Miscellaneous								
Keyword	**Application**	**S**	**C**	**F**	**Sub**	**Util**	**CLI**	**Chapter**
ABS	absolute value function.			•				3
AND	logical AND function.			•				3
ASC	extract a number.			•				3
ATTACH	attach to a de-attached job.						•	8
BUILD	build a command file.						•	8
BYE	log off the system.	•	•					3
ASG	assign a device.						•	8
BLDCOM	build documentation file.						•	8
CDIR	create a directory.						•	8
CPART	create a partition.						•	8
CHR$	make a string of the value of a number.			•				3
CLI	execute the BASIC CLI.					•		7
CON	continue execution of a program.		•					3
CRM$	cram a string.			•				3
DIR	change to another directory.						•	8

Licensed Material-Property of Data General Corporation

Table 2-1. Keyword Summary (continued)

Miscellaneous (continued)								
Keyword	Application	S	C	F	Sub	Util	CLI	Chapter
DISK	display disk information.						•	8
DIV.SL	calculate extended percentage.				•			7
EQUIV	rename a device.						•	8
ERM$	error message function.			•				3
FILL$	fill a string.			•				3
GETCM.SL	read CLI command string.				•			7
GDIR	display directory name.						•	8
GSDIR	display system directory name.						•	8
GSYS	display system name.						•	8
GTOD	display time and date.						•	8
INIT	initialize a device.						•	8
INT	integer function.			•				3
LEN	string length function.			•				3
LSPEED	set terminal speed.					•		7
MAX	maximum of two numbers.			•				3
MDIR	display master directory name.						•	8
MIN	minimum of two numbers.			•				3
MOD	remainder after dividing two numbers.			•				3
MSG	send a message to a terminal.		•					3
MUL.SL	calculate percent of a number.				•			7
OR	logical OR function.			•				3
PAGE	set the page width.		•					3
POP	exit the BASIC CLI.						•	8
PORTS	display terminal assignments.					•		7
QUIT	exit the BASIC CLI.						•	8
RANDOMIZE	reset the random number generator.	•	•					3
PRTCOM	print documentation file.						•	8
FREE	free a device.						•	8
RELEASE	release a device.						•	8

Licensed Material-Property of Data General Corporation

S	Statement
C	Command
F	Function
Sub	Subroutine
Util	Utility
CLI	CLI Command

Table 2-1. Keyword Summary (continued)

Miscellaneous (continued)								
Keyword	**Application**	**S**	**C**	**F**	**Sub**	**Util**	**CLI**	**Chapter**
RENUMBER	renumber program lines.		•					3
RND	random number generator.			•				3
SDIR	set the system directory.						•	8
SGN	sign function.			•				3
SHFT	logical shift function.			•				3
SLINE	select a line and attach.						•	8
SPDIS	disable spooling.						•	8
SPEBL	enable spooling.						•	8
SPKILL	kill spooling.						•	8
SQR	square root function.			•				3
START	start a de-attached job.						•	8
STAT	system status.						•	8
SYS	system information.			•				3
TAB	set tab zones for output.	•	•					3
TERM	set certain terminal keys.					•		7
TRUN$	truncate a string.			•				3
UCM$	uncram a string.			•				3
VAL.SL	convert string to number.				•			7

Licensed Material-Property of Data General Corporation

S	Statement
C	Command
F	Function
Sub	Subroutine
Util	Utility
CLI	CLI Command

Table 2-1. Keyword Summary (continued)

INFOS								
Keyword	**Application**	**S**	**C**	**F**	**Sub**	**Util**	**CLI**	**Chapter**
CLOSE	close an INFOS file.	•						9
DFSI	define a subindex.	•						9
DELETE	delete an INFOS record.	•						9
DELSI	delete a subindex.	•						9
LNKSI	link to a subindex.	•						9
OPEN	open an INFOS file.	•						9
READ	read an INFOS record.	•						9
REINS	reinstate a deleted record.	•						9
RETDF	retrieve a subindex definition.	•						9
RETHK	retrieve highest key in index.	•						9
RETKY	retrieve a key in index.	•						9
RETST	retrieve record status.	•						9
REWRITE	rewrite an INFOS record.	•						9
WRITE	write an INFOS record.	•						9

End of Chapter

Licensed Material-Property of Data General Corporation

Chapter 3
Statements, Commands, and Functions

This chapter describes the following statements, commands, and functions in alphabetical order ("F" stands for function, "S" for statement, and "C" for command):

Table 3-1. Summary of Statements, Commands, and Functions

Keyword	Type	Description
ABS	F	Absolute value of a number.
AND	F	Logical AND of two expressions.
ASC	F	Extract a number from a string.
BLOCK READ	S, C	Block input from file or common area.
BLOCK WRITE	S, C	Block output to file or common area.
BYE	S, C	Log off the system.
CHAIN	S, C	Execute a utility or another program.
CHR$	F	Make a string of a number.
CLOSE FILE	S, C	Close any or all open files.
CON	C	Continue execution of a stopped program.
CRM$	F	Cram 3 bytes of string into 2 bytes.
DATA	S	Data for variables in READ statement.
DEF	S	User defined function statement.
DELAY	S, C	Delay execution of a job.
DELETE	S, C	Delete a file from your directory.
DIM	S, C	Dimension arrays and string variables.
END	S	Optional program terminator.
ENTER	S, C	Compile character formatted program source statements.
EOF	F	End-of-file test.
ERASE	S, C	Erase lines in working storage.
ERM$	F	String for error messages.
FILL$	F	Fill a string with an ASCII value.
FOR/NEXT	S	Define a program loop.
GOSUB/RETURN	S	Go to and return from a program subroutine.
GOTO	S	Go to a specific statement.
GPOS	F	Get the file pointer's position.
IF	S	Conditional transfer or execution.
INPUT *[FILE]*	S, C	Input characters from file or terminal.

Licensed Material-Property of Data General Corporation

Table 3-1. Summary of Statements, Commands, and Functions (continued)

Keyword	Type	Description
INPUT *[FILE]* USING	S, C	Input allowing error trap and three alternate terminators.
INT	F	Integer function.
KADD	S, C	Add a key to an index.
KDEL	S, C	Delete a key from an index.
KFIND	S, C	Find a key in an index.
KNEXT	S, C	Find the next key in an index.
LEN	F	Length of a string.
LET	S, C	Assign a value to a variable.
LIST	C	Display contents of working storage or LIST contents to a character format file.
LOAD	C	Bring into working storage a SAVEd or REPLACEd program.
LOCK/UNLOCK	S, C	Lock and unlock blocks of a file.
MAX	F	The maximum of two values.
MIN	F	The minimum of two values.
MOD	F	The remainder after dividing two values.
MSG	C	Send a message to another terminal.
MTDIO	S, C	Direct mag tape I/O.
NEW	S, C	Clear working storage.
ON ERR	S	Trap an error in your program.
ON GOTO or GOSUB	S	Conditional transfer to statement or subroutine.
ON IKEY	S	Trap an interrupt in your program.
OPEN FILE	S, C	Open a file for access.
OR	F	Logical inclusive OR of two expressions.
PAGE	C	Set the page width of your terminal.

Licensed Material-Property of Data General Corporation

Table 3-1. Summary of Statements, Commands, and Functions (continued)

Keyword	Type	Description
POSITION FILE	S, C	Position to a record or a byte in a file.
PRINT *[FILE]*	S, C	Print characters to a file or terminal.
PRINT *[FILE]* USING	S, C	Print using a specified format.
RANDOMIZE	S, C	Reseed the random number generator.
READ	S	Get values in DATA for variables.
READ FILE	S, C	Input binary data from a file.
REM	S	Comment in your program.
RENAME	C	Rename a file.
RENUMBER	C	Renumber the line numbers or current program.
REPLACE	S, C	Replace a SAVEd program with another program.
RESTORE	S, C	Reset the data list pointer in DATA.
RND	F	Random number generator.
RUN	C	Execute a program.

Keyword	Type	Description
SAVE	S, C	Save a program on disk in a program file.
SGN	F	Sign of a number.
SHFT	F	Logical shift of an expression.
SIZE	C	Determine the size of current program
SQR	F	Square root of a number.
STMA	S, C	User system calls.
STOP	S	Stop program execution.
SWAP	S, C	Execute another program and return.
SYS	F	System information.
TAB	S, C	Set width of tabulated output zones on a terminal.
TINPUT *[FILE]*	S, C	Timed input
TRUN$	F	Truncates a string.
UCM$	F	Uncrams a crammed string (see CRM$).
WRITE FILE	S, C	Output binary data to a file.

Licensed Material-Property of Data General Corporation

Statement
Command
● *Function*

ABS

Absolute value of a number.

Format

ABS (expr)

Arguments

expr a numeric expression.

Why Use It?

Use ABS to get the absolute (positive) value of a negative or positive number.

What It Does

ABS does nothing to a positive number. ABS strips the negative sign from a negative number making it positive.

How to Use It

A numeric expression can be a number; a single, double, or triple precision variable (if you're using triple precision) assigned a numeric value; an array element; a function; or a combination of any of these in a numeric expression with arithmetic operators. You can use the ABS function as a numeric expression wherever numeric expressions are allowed.

Examples

```
*PRINT ABS(-30)
 30
*

0010 INPUT X,Y
0020 FOR I=ABS(X) TO ABS(Y)
.
.
.
```


Licensed Material-Property of Data General Corporation

AND

Logical AND of two expressions.

Format

AND ($expr_1$, $expr_2$)

Arguments

$expr_1$ and $expr_2$	numeric expressions or variables you want compared.

Why Use It?

AND is frequently used to check if a bit is set to 1 in a single, double, or triple precision number.

What It Does

To illustrate how AND checks a bit, we line up the bits for both exprs in Figure 3-1. For each bit, the result is 1 only if the corresponding bits in both exprs are 1; otherwise the result is 0.

How to Use It

If, for example, a status flag was returned to you, and you needed to check bit location 6, you'd AND the status flag with the value 64, as we did in Figure 3-1.

The value 64 is 2 to the 6th power, so if you want to check bit *n*, you'd AND the value with 2 to the *nth* power. See Figure 3-2 for a chart of the powers of 2.

If you want to check several bits at once, add the bit values together and AND the total with your expr. For example, to check bits 6 and 7 at once, we'd add 64 (bit 6) and 128 (bit 7) to get a value of 192 to AND with expr. You can use AND as a numeric expression wherever numeric expressions are allowed.

Examples

```
*PRINT AND(192,64) ) 64
*
```

```
.
0030 MTDIO 1,1,REC$,ER%

0040 IF AND(ER%,256) THEN GOTO 0300
.
.
.
```

In this example, we first read a tape record into REC$, and ER% holds the tape status. If the AND of ER% and 256 produces a value other than 0 (this would only happen if bit 8 is set in ER%), go to the end-of-file routine at line 300.

AND (192,64)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	BIT POSITION
0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	$expr_1$ = 192
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	$expr_2$ = 64
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	RESULT = 64(2^6)

SD-01175

Licensed Material-Property of Data General Corporation

BIT POSITION	n	2↑n		2^n
10000000000000000000000000000000	31	2↑31	or	2147483648
01000000000000000000000000000000	30	2↑30	or	1073741824
00100000000000000000000000000000	29	2↑29	or	536870912
00010000000000000000000000000000	28	2↑28	or	268435456
00001000000000000000000000000000	27	2↑27	or	134217728
00000100000000000000000000000000	26	2↑26	or	67108864
00000010000000000000000000000000	25	2↑25	or	33554432
00000001000000000000000000000000	24	2↑24	or	16777216
00000000100000000000000000000000	23	2↑23	or	8388608
00000000010000000000000000000000	22	2↑22	or	4194304
00000000001000000000000000000000	21	2↑21	or	2097152
00000000000100000000000000000000	20	2↑20	or	1048576
00000000000010000000000000000000	19	2↑19	or	524288
00000000000001000000000000000000	18	2↑18	or	262144
00000000000000100000000000000000	17	2↑17	or	131072
00000000000000010000000000000000	16	2↑16	or	65536
00000000000000001000000000000000	15	2↑15	or	32768
00000000000000000100000000000000	14	2↑14	or	16384
00000000000000000010000000000000	13	2↑13	or	8192
00000000000000000001000000000000	12	2↑12	or	4096
00000000000000000000100000000000	11	2↑11	or	2048
00000000000000000000010000000000	10	2↑10	or	1024
00000000000000000000001000000000	9	2↑9	or	512
00000000000000000000000100000000	8	2↑8	or	256
00000000000000000000000010000000	7	2↑7	or	128
00000000000000000000000001000000	6	2↑6	or	64
00000000000000000000000000100000	5	2↑5	or	32
00000000000000000000000000010000	4	2↑4	or	16
00000000000000000000000000001000	3	2↑3	or	8
00000000000000000000000000000100	2	2↑2	or	4
00000000000000000000000000000010	1	2↑1	or	2
00000000000000000000000000000001	0	2↑0	or	1

SD-01176

Figure 3-2. The Powers of 2

Licensed Material-Property of Data General Corporation

ASC

Extract a number from a string.

Format

var = ASC(strvar)

Arguments

var numeric variable to receive 4 bytes or less (6 or less if you have triple precision numbers) it can be an array element.

strvar string variable or substring.

Why Use It?

It is sometimes more efficient to pass data to or from files or the common area as strings. If you use the CHR$ function to put a number into a string, you should use the ASC function to extract the number from that string.

What It Does

ASC takes the right-most 32 bits (16 bits if var is single precision, 48 bits if var is triple precision) of strvar and puts them into var. A straight byte transfer takes place. You get the binary value of whatever is in strvar. If strvar contained an "A" only, you'd get the value 65 in var, because 65 is the decimal ASCII code for "A". Since the CHR$ function puts the number into a string as a binary value, the ASC function would return the binary value of the number itself.

How To Use It

Use the ASC function in assignment statements (or the LET command) only, because you need to assign the bytes to var. You can assign 4 bytes of strvar to a regular 4-byte numeric variable, 2 bytes of strvar to a single precision variable, or 6 bytes to a triple precision variable. However, if you assign more than 2 bytes to a single precision variable, only 2 bytes will be transferred when you perform any I/O with the single precision variable.

Examples

```
0010 DIM A$(12)
0020 INPUT B
0030 LET A$(1,4)=CHR$(B,4)
.
.
. 0010 LET B=ASC (A$(1,4))
.
.
.
```

Line 30 makes 4-byte string of B, and line 100 extracts B from A$.

```
*DIM A$(4)
*LET A$="A"
*LET VAL=ASC(A$)
*PRINT VAL   65
*
```


Licensed Material-Property of Data General Corporation

- *Statement*
- *Command*
  Function

BLOCK READ

Input blocks from file or common area.

Format

BLOCK READ *[FILE (channel,start),]* var

Arguments

var — a string variable DIMensioned to at least 512 bytes, or an array variable for an array that holds at least 128 elements.

channel — channel number of a file expressed as a number or numeric expression.

start — block number from which to start inputting, if inputting from a file, expressed as a number or numeric expression.

Why Use It?

Use BLOCK READ to input data from the common area, and BLOCK READ FILE to input data from a file. BLOCK statements transfer only 512-byte blocks at a time. Whenever you BLOCK READ from a file or from a common area, you read no less than 512 bytes into your var. Block I/O is discussed in Chapter 4 in the section, "File Access"; you should read that section before trying to use BLOCK statements.

What It Does

BLOCK READ FILE will read -- as many blocks (512-byte sectors) as the string variable or array can hold -- from a file into a string or array variable. If string variable A$ was DIMensioned to 1536 bytes, BLOCK READ FILE would read 3 blocks into it -- but if A$ was only 1535 bytes long, only 2 entire blocks would fit. BLOCK READ FILE will start reading blocks sequentially from the block number specified by *start*. The byte position in the file (GPOS) after a BLOCK READ FILE is undefined. If var is larger than the amount of data in the 512-byte block transferred, var will be padded with nulls.

BLOCK READ (i.e., without the optional keyword FILE) will read only one 512-byte block from the common area. (You cannot specify *start* and *channel* without FILE.) There is a common area for each job (i.e., for each terminal) which is like a file that is always open and ready for use; it is only one block (512 bytes) long. Use the common area to pass information to and from programs when you do SWAPs and CHAINs.

How To Use It

To read blocks from a file, you must first open the file and assign a channel number to the file. If your channel number is 16 (16 is a dummy channel referring to your terminal), BLOCK READ FILE will behave exactly as a simple BLOCK READ and refer to the common area. Otherwise, BLOCK READ FILE will read from the file specified by *channel*. You must specify the block number at which to start reading in *start*. BLOCK READ FILE will start reading sequentially from that point.

The size of var determines the number of blocks you want transferred. A block is 512 bytes. String variables must be DIMensioned to at least 512 bytes; if you make them larger, BLOCK READ FILE will read in as many entire blocks as the string variable will hold, padding the unused portion of the string variable with nulls. The same is true for arrays, but array elements hold 4 bytes each, so in this case, you need a 128-element array to hold 512 bytes (1 block), a 256-element array to hold 1024 bytes (2 blocks), etc.

BLOCK READ (i.e., without FILE), or BLOCK READ FILE (16) will refer to the common area. Use the common area to pass information to a program you are SWAPping to, CHAINing, or RUNning. If you send information into the common area with a BLOCK WRITE, any other program can pick up the information with a BLOCK READ, unless another BLOCK WRITE or a log-off overwrites your common area.

Licensed Material-Property of Data General Corporation

Examples

```
*LIST )
0010 REM -- READ COMMON INTO A$
0020 DIM A$(512)
0030 BLOCK READ A$
0040 PRINT LEN(A$)

*RUN )
512
*

0010 REM -- READ 2 BLOCKS FROM "INFO" INTO ARRAY
0020 DIM AR(3,63)
0030 OPEN FILE (0,0)"INFO"
0040 BLOCK READ FILE (0,0),AR
.
.
.
```

Line 40 starts at block 0 in file 0 and reads 2 blocks into AR.

Array AR is 4 times 64, or 256 elements.

Licensed Material-Property of Data General Corporation

- *Statement*
- *Command*
  Function

BLOCK WRITE

Output blocks to file or common area.

Format

BLOCK WRITE *[FILE (channel,start),]* var

Arguments

var — a string variable DIMensioned to at least 512 bytes, or the array name of an array that holds at least 128 elements.

channel — channel number of a file, expressed as a number or numeric expression.

start — the number of the first block in the file to receive output, expressed as a number or numeric expression.

Why Use It?

Use BLOCK WRITE to write data to the one block (512-byte) common area, and BLOCK WRITE FILE to write blocks of data to a file. BLOCK statements only transfer 512-byte blocks at a time: whenever you BLOCK WRITE to a file or to the common area, you write at least 512 bytes of data. Block I/O is discussed in Chapter 4 in the section called "File Access"; you should read that section before trying to use BLOCK statements.

What It Does

BLOCK WRITE FILE will write the number of 512-byte blocks needed to contain the data in var to a file specified by *channel* starting at the block specified by *start*. If var is bigger than 512 bytes but less than or equal to 1024 bytes, 2 entire blocks (1024 bytes) are written to the file and padded with nulls. If var is 1025 bytes, 3 blocks are written to the file and padded with nulls up to 1536 bytes. You can start at any block number and write sequentially from that point. The file position pointer (GPOS) is undefined after a BLOCK WRITE FILE statement.

BLOCK WRITE (i.e., without FILE) or a BLOCK WRITE FILE (16) (where 16 is a dummy channel for your terminal) will output the contents of var to a special one block (512-byte) file called a common area. Since the common area will only hold 512 bytes, you can write only one block at a time.

How To Use It

To write blocks to a file (except the common area), you must open the file and assign a channel number to the file, so that you can use the channel number in *channel*.

If your channel number is 16 (dummy channel for your terminal), BLOCK WRITE FILE will behave like a BLOCK WRITE and refer to the common area. Otherwise, BLOCK WRITE FILE will write to a file and you have to specify what block number at which the write operation should begin in *start*.

The size of var determines the number of blocks you want written, so if you want to write 2 blocks you should DIMension var to hold more than one block and less than or equal to 2 blocks. If var is even 1 byte longer than 2 blocks, 3 blocks will be written and padded with nulls.

If you want to use the common area for your job, use BLOCK WRITE or BLOCK WRITE FILE (16) to fill the common area with data, and use BLOCK READ or BLOCK READ FILE (16) to retrieve the data. You can use the common area to pass information to a program you are SWAPping to, CHAINing, or RUNning. The information you send to the common area with a BLOCK WRITE will stay in the common area until you do another BLOCK WRITE to the common area or until you log off. The common area is always open and ready for use.

Examples

```
0005 REM -- WRITE X$ TO COMMON
0007 REM -- WHERE OPEN CAN FIND IT.
0010 DIM X$(512)
0020 LET X$="SUB1,5,SUB2,5,PHYS,6",FILL$(0)
0030 BLOCK WRITE X$
0040 SWAP "OPEN"

0050 BLOCK READ X$

.
.
.
```

Line 30 sends X$ into common, where OPEN will find it. OPEN uses a BLOCK READ to read X$, then outputs a new X$ back into common with a BLOCK WRITE. Line 50 reads the new X$.

```
0010 REM -- WRITE ARRAY
0015 REM -- TO 2 BLOCKS OF "INFO".
0020 DIM AR(3,63)
.
.       (Code to fill array goes here.)
.
0080 OPEN FILE (0,0),"INFO"
0090 BLOCK WRITE FILE (0,2),AR
```

Line 90 starts at block 2 of INFO, and writes the entire array to 2 blocks of the file. No matter how much of the array contains actual data, 2 entire blocks are written and padded with nulls if necessary.

Licensed Material-Property of Data General Corporation

- *Statement*
- *Command*
- *Function*

BYE

Log off the system.

Format

BYE

Why Use It?

Use BYE when you want to run the normal log-off procedure.

What It Does

BYE will terminate a Business BASIC session with some information (set by your system manager).

How To Use It

Just type BYE as a keyboard mode command, or preceding with a line number as a program statement. See Chapter 2 for a description of logging on, if you want to log on to the system again.

Examples

```
*BYE )

*0200 BYE )
```

- *Statement*
- *Command*
- *Function*

CHAIN

Execute a utility or another program.

Format

CHAIN { strvar$ / "filename" } *[THEN* { *GOTO linenum* / *CON* }]

Arguments

filename a literal filename in quotes for a save file on disk, *not* an ASCII listing or source file. The file must contain a BASIC program or Business BASIC utility program.

strvar$ a string variable you've already DIMensioned and assigned the value of filename.

linenum a valid line number in the program you are CHAINing to. Execution will begin at *linenum* rather than the beginning of the program, and the program retains the values it had when it was SAVEd.

Why Use It?

Use CHAIN to execute another program or a Business BASIC utility program from the program which is running. The current running program is cleared from working storage and the new program is executed. Use SWAP instead of CHAIN if you want to return to the original program (see SWAP in this chapter).

Licensed Material-Property of Data General Corporation

CHAIN (continued)

What It Does

CHAIN searches your directory for filename; if the file is not found, it searches the library directory. If the new program is found, your currently running program is cleared from working storage and the new program is loaded into your working storage. If the new program is not found, your currently running program remains in working storage and you get an error message. (The BASIC CLI CHAIN command works the same way -- see CHAIN in Chapter 8.)

The CHAINed program runs, by default, from the lowest line number in the program, and all variables are cleared as if a RUN had occurred (see RUN, this chapter). If you specify THEN GOTO *linenum* or THEN CON in a CHAIN statement, then it acts as if a CON or RUN *linenum* occurred, and all variables retain the values they had when the program was SAVEd (see CON and/or RUN in this chapter). CHAIN does not change the status of files. Open files remain open, and current file position pointers are maintained.

How To Use It

CHAIN can be a program statement or a keyboard mode command. To start the new program at the beginning, clear all variables, etc., and use CHAIN without THEN CON or THEN GOTO. If you want to resume execution of the new program from its point of interruption, use CHAIN THEN CON. If you want to start execution at a certain line number, use CHAIN THEN GOTO *linenum*. If you had wanted the new program to execute and then return you to the original calling program, you'd use SWAP instead of CHAIN (see SWAP in this chapter).

Examples

0090 CHAIN "PROG102"

Executes program "PROG102".

```
*CHAIN "SCRATCH" THEN CON )
```

Executes "SCRATCH" from where it last stopped.

```
0010 DIM NAME$(10)
0020 LET NAME$="PROG3"
0030 CHAIN NAME$ THEN GOTO 0100
```

Executes "PROG3" starting at line 100.

CHR$

Put binary value of number into a string.

Format

strvar = CHR$(expr *[,byte]*)

Arguments

strvar string variable or substring to receive bytes containing the binary value of expr.

expr a numeric expression, numeric variable, or array element.

byte optional number of bytes, that can be up to and including 6. If you do not include *byte*, a 1-byte string is made.

Why Use It?

Use the CHR$ function to put the binary value of a number into a string. It is sometimes more efficient (and sometimes required) that you pass data as a string to and from files or the common area. Although you can say LET A$=123, this assignment creates a 3-byte string with the digits 1, 2, and 3 as characters (ASCII values) in the string. The CHR$ function would put the binary value of 123 into the string.

What It Does

The CHR$ function takes the binary value of expr and puts it in the first *byte* bytes of strvar. *Byte* can be any number from 1 to 6; if you don't specify *byte*, 1 is assumed.

How To Use It

You use the CHR$ function in an assignment statement (or LET command) only, because you have to assign it to strvar. Strvar can be a string variable or substring. If you make a string of 6 bytes, make sure you use a triple precision variable to extract the same number of bytes with ASC when you want the number back. Make a 4-byte string for double and single precision numbers.

Licensed Material-Property of Data General Corporation

Examples

0080 LET D$=CHR$(C%,2),CHR$(0,4),CHR$(R1*50,4), CHR$(0,2)

This statement will make a string concatenated to 2 bytes of C%, 4 bytes of 0, 4 bytes of the value R1*50, and 2 bytes of the value 0.

```
*DIM X$(1) )
*LET X=65 )
*LET X$=CHR$(X) )
*PRINT X$ ) A
*DIM Y$(4) )
*LET Y$=CHR$(5,4) )
*LET VAL=ASC(Y$) )
*PRINT VAL ) 5
*
```


Licensed Material-Property of Data General Corporation

- *Statement*
- *Command*
  Function

CLOSE

Close all opened files or a specific file.

Format

CLOSE *[FILE (channel)]*

Arguments

channel channel number of a specific open file you want closed. If *channel* is 16, it refers to all open files at your terminal, and is equivalent to a simple CLOSE.

Why Use It?

Use CLOSE [FILE] to disassociate a filename and a channel so that the file can no longer be referred to using *channel.* Close a file when you've finished using the file, or if you want to reOPEN the file in a different mode. Close all files when your program that uses the files is finished.

What It Does

CLOSE by itself closes all open files, freeing their channel numbers so that you can use them again for other files or for the same file. CLOSE FILE with *channel* will close the file specified by *channel* and free *channel* for you to use again. If *channel* is 16, CLOSE FILE refers to the terminal (job) and closes any open files at that terminal.

How To Use It

Most programs that access files close the files when they are finished using them, or at the end of the program. If you forgot to close a file and you try to OPEN it, you'll get an error message that says the file is already opened. You can then issue a CLOSE or CLOSE FILE command while in keyboard mode, and then proceed to OPEN the file.

Examples

```
0010 REM -- OPEN "DATA" ON 2
0020 OPEN FILE (2,0), "DATA"
0030 REM -- PROCESS FILE
.
.
.
0490 REM -- CLOSE "DATA"
0500 CLOSE FILE (2)
.
.
.
0999 REM -- CLOSE ALL FILES
1000 CLOSE
```

If an interrupt, error, or STOP occurred before line 500 above, and you tried to RUN the program from the top (yet the CLOSE statement at line 500 had not been executed), you'd get:

ERROR 42 -- FILE ALREADY OPENED

because you are trying to execute line 20 again. You then type the command:

```
*CLOSE )
```

or

```
*CLOSE FILE (2) )
```

and resume execution of the program.

Licensed Material-Property of Data General Corporation

Statement
• *Command*
Function

CON

Continue execution of a stopped program.

Format

CON

Why Use It?

Use CON to continue execution of a program after it stopped as a result of a STOP statement, an interruption, or an error condition. CON is equivalent to RUN *linenum* (see RUN, this chapter) where *linenum* is the line number of the statement at which you want to start execution -- i.e., usually the statement following the STOP statement or the point of interruption or error. This is useful for debugging a program, since you can resume execution if an error or interrupt occurred and still retain the variable assignments and file status of files in the program.

What It Does

CON continues the execution of a program from the point where it last stopped. CON will not affect values of variables or file status of files (i.e., open files remain open, file position pointers stay where they are, locks are still active).

How To Use It

Use CON as a keyboard mode command only. If your program stops for any reason other than it being finished, you can edit it, test variables for their values, etc., then type CON to resume execution of the program.

Example

```
*RUN )
.
.

(interrupt or error occurs)

*PRINT X,Y,Z ) 12  14  123

(Test values)

*10 LET X=13 )
*LET X=13 )

(Rewrite statement 10 to correct program, and set X
equal to 13 to resume execution.)

*CON )
.
.
.
```


Licensed Material-Property of Data General Corporation

CRM$

Cram every three bytes of a string into two bytes.

Format

$strvar_1$ = CRM$ (strvar)

Arguments

$strvar_1$ string variable or substring to receive crammed string.

strvar string variable or substring you want crammed, may be the same as $strvar_1$.

Why Use It?

Use the CRM$ function to shorten the amount of space needed to hold a string, and use the UCM$ function to uncram the string for printing and other purposes. Index files allow you to use crammed keys as entries (File Maintenance does it for you).

What It Does

Every three bytes of strvar are crammed into two bytes of $strvar_1$. You can only cram the following 40 characters: A to Z, 0 to 9, and four special characters set either by STMA 10 or the default characters -- comma (,), decimal point (.), minus sign (-), and space. You cannot cram lowercase characters. Any character in strvar not in this set is converted to the zero value character, and when $strvar_1$ is uncrammed, the zero value character becomes a space.

How To Use It

You can only use the CRM$ function in assignment statements (or the LET command) because you have to assign the three bytes to two bytes of a string. You can make $strvar_1$ and strvar the same string variable. You must use UCM$ on a crammed string to print the string or perform other string functions on it.

Examples

```
0010 DIM X$(6),Y$(9)
0020 LET Y$="ABCDEFGHI"
0030 LET X$=CRM$(Y$)
0040 LET Y$(1,6)=X$
0050 LET Y$=UCM$(X$)
0060 PRINT Y$
*RUN )
ABCDEFGHI
*

*DIM X$(6) )
*LET X$="SIXCHR" )
*PRINT LEN(X$) )
6
*LET X$=CRM$(X$) )
*PRINT LEN(X$) )
4
*
```


Licensed Material-Property of Data General Corporation

- *Statement*
  Command
  Function

DATA

Data for variables in READ statements.

Format

DATA value *[,value ...]*

Arguments

value numeric constants, or string literals in quotes. You can use any combination of them, but no variables.

Why Use It?

Use DATA statements in conjunction with READ statements to provide values for variables quickly and easily (see READ, this chapter). You can assign, for example, ten values to ten variables in one READ and one DATA statement, and you can change those values merely by changing the one DATA statement.

What It Does

The values in all DATA statements in a program form a single data list. The first value in the list is the first value in the lowest numbered DATA statement, and the last *value* in the list is the last *value* in the highest numbered DATA statement. Each variable in the variable list of a READ statement picks up a value from the DATA list, starting in sequence: the first variable in a READ picks up the first value in the data list, then the next variable picks up the next *value*, etc. You can make a READ statement start at the beginning of the data list by doing a RESTORE (see RESTORE, this chapter).

How To Use It

You can only use DATA as a program statement. You can have many DATA statements in a program, and many values of mixed types (numeric and string) in a single DATA statement, separated by commas. DATA statements can appear anywhere in a program, even after STOP statements, because the system doesn't execute them. You cannot use DATA after THEN in IF/THEN, ON ERR, or ON IKEY statements.

Examples

```
0010 READ X,Y,A$,B$,Z
0020 PRINT X,Y,A$,B$,Z
```

X will equal 2, Y will equal 4, A$ will equal "HELLO," B$ will equal "GOODBYE", and Z will equal 20.

```
0050 STOP
0060 DATA 2,4,"HELLO","GOODBYE",4*5

*RUN )

 2  4  HELLO  GOODBYE  20

STOP AT 0050
*
```


Licensed Material-Property of Data General Corporation

• *Statement*
Command
Function

DEF

Define your own function.

Format

DEF FNa(d) = expression

Arguments

a	a single letter in the range A to Z that serves as the function name with the FN prefix.
d	a dummy variable to be replaced with a value when you reference the function.
expression	a numeric expression for the calculation you wish to perform. You cannot use strings and string expressions here.

Why Use It?

DEF allows you to define a function to which you can refer in a program without having to repeat the function. They are called *user-defined functions.*

What It Does

When you refer to a defined function by its FN-name, the value you supply for the dummy variable d is used in the function's calculations.

How To Use It

When you define a function, you create a name for it: use a single letter a with the prefix FN. You also supply a dummy variable d in parentheses, and a numeric expression for expression. Your expression is the calculation you want your function to perform each time you refer to it. The variable you supply for d can also be used in expression.

When you refer to the function by its FN-name, you supply a value for d (e.g., FNA(12) or FNB(X+Y)). This value replaces d in your expression, and the function performs the calculations using this value, thereby giving you the desired result. If d is not in expression, the value you supply will not matter, but any variables you use in expression will be evaluated and their values will be used in the calculations.

You can use a function's FN-name with a supplied value for d in any statement in which you can have numeric expressions. Your expression can include one or more supplied or previously defined functions. This is called *nesting,* and you can nest user-defined functions to a depth of four. Any and all functions defined in a program or in a keyboard mode command only exist for the current program in working storage. DEF limits you to single-line formulas. For longer formulas, use subroutines.

Examples

```
0010 LET X=128
0020 LET Y=50
0030 DEF FNA(D)=D/X*Y
0040 DEF FNB(D)=X/Y*FNA(X)
0050 LET VAL=FNA(20)
0060 LET TOTAL=FNB(0)
.
.
.
```

Line 30 defines function FNA, and line 40 defines a nested function FNB. Line 50 uses FNA, and line 60 uses FNB (0 is a dummy value not used).

Licensed Material-Property of Data General Corporation

- *Statement*
- *Command*
- *Function*

DELAY

Delay execution of the next program statement or command.

Format

DELAY expr

Arguments

expr a numeric expression or a variable with an assigned numeric value.

Why Use It?

Use DELAY to delay the execution of the following statement in a program, or to delay return of the keyboard mode prompt.

What It Does

DELAY delays the execution of the following program statement. If DELAY is the first statement of your program, DELAY will delay the entire program. If DELAY is a keyboard mode command, it will delay return of the prompt and not permit any typing on the terminal (except use of the interrupt key).

How To Use It

You supply expr to define how many tenths of a second the computer should wait. If you hit the interrupt key during a DELAY, you will stop the delay and return to keyboard mode.

Examples

0010 DELAY 20

This statement causes a 2-second delay.

```
*DELAY 30 )
```

This delays the program in working storage for 3 seconds.

- *Statement*
- *Command*
- *Function*

DELETE

Remove a file from your directory.

Format

DELETE { strvar | "filename" }

Arguments

filename a string literal in quotes that is the name of a file in your directory.

strvar a string variable that represents filename.

Why Use It?

Use DELETE to remove the disk file named filename from your directory, effectively deleting the file from disk.

What It Does

DELETE searches your directory for filename and erases filename and the file itself, freeing up space formerly used by the file.

How To Use It

If you use strvar for filename, it must already be DIMensioned and assigned a value of filename. You can't interrupt an execution of DELETE, so be careful!

Examples

```
*DELETE "FILE101.LS" )
```

```
0080 DIM NAME$(10)
0090 INPUT NAME$
0100 DELETE NAME$
.
.
.
0700 DELETE "KERMIT"
```


Licensed Material-Property of Data General Corporation

- *Statement*
- *Command*

  Function

DIM

Dimension arrays and/or strings.

Format

DIM { svar$(n) / array(col) / array(row,col) } [{ ,*svar$(n)* / ,*array(col)* / ,*array(row,col)* } ...]

Arguments

array is a variable name for an array, that can be one to six characters.

svar$ is a string variable name ending with a $ symbol.

n is the maximum string length of svar$. It can be a variable or a numeric expression.

col is the number of elements in a one-dimensional array. It can be a numeric expression or a variable.

row is the number of rows in a two-dimensional array. It can be a numeric expression or a variable.

Why Use It?

To set the dimensions of one-dimensional or two-dimensional arrays and/or to set the maximum length for string variables. You can dimension arrays and string variables in one DIM statement.

What It Does

The information you supply in a DIM statement or command is used to allocate storage. The value of col defines how many column positions a one-dimensional array will need, with one element per column position. If you specify row and col, these values will determine how many rows and column positions a two-dimensional array will need. Both one-dimensional and two-dimensional arrays start at column position 0; a one-dimensional array with 16 elements would start at column position 0 and end at column position 15, and would be dimensioned using "15" as the value of col.

If you use an array but do not declare it in a DIM statement or command, Business BASIC sets aside 11 elements (column positions 0 to 10) for a one-dimensional array, and as a default, 121 elements (columns 0 to 10, rows 0 to 10) for a two-dimensional array. An undeclared one-dimensional array can't have more than 11 elements, and an undeclared two-dimensional array can't have more than 121 elements; if you need more space, dimension the space using DIM. There are no restrictions on the number of elements you can place in an array other than restrictions due to available core.

You must dimension string variables; there is no default. The value of n will define how many characters you can have in the string variable. String variables begin with 1, *not* 0, so A$(10) will definitely have 10 characters. There are no restrictions on the number of characters you can place in a string variable other than restrictions due to available core.

How To Use It

In the same DIM statement you can DIMension an array and a string variable in any order. You can also DIMension several arrays and string variables, or use separate DIM statements. If you only specify col for array, you will dimension a one-dimensional array. If you specify both row and col, you will dimension a two-dimensional array. You can use parentheses or square brackets to enclose your subscripts. The values for row and col can be expressed as variables or numeric expressions; they must evaluate to integers, and if you use variables, they must have pre-assigned values. Your variable for array must follow the rules for Business BASIC variables (see Chapter 2).

Your svar$ must follow the rules governing string variables (see Chapter 2). The value you supply for n (string length) can be expressed as a variable or a numeric expression, but it must evaluate to an integer. Again, you can use parentheses or square brackets to enclose your subscripts.

You can refer to any element of an array or any character of a string variable by using the name of the array or the string variable name and supplying the identifying subscript. For examples, see Chapter 2. Remember, arrays begin at 0, so if you want the first element of the first row of two-dimensional array C, you have to ask for C(0,0), *not* C(1,1). On the other hand, string variables start at 1, *not* 0. To get the third character in a string variable you can ask for A$(3) and get it.

Licensed Material-Property of Data General Corporation

DIM (continued)

You can use DIM as a program statement or a keyboard mode command, but if you use it as a command, it will only apply to a program already executing (i.e., before using CON), because a RUN will reset all program variables. In either case, if you use subscripted variables in a DIM statement or command, you must supply values for the variables before executing DIM.

You can redimension a previously dimensioned array or string variable during execution of a program, but you will not alter the size of the storage space already set aside for them. You can only redimension arrays and string variables to the same number or fewer elements; if fewer, you will not be able to use or refer to the extra locations. Redimensioning is used primarily to change the subscipts of two-dimensional arrays.

Examples

0010 DIM ARRAY(5,6), C(20), STRING$(30)

ARRAY is a 6x7 element two-dimensional array. C is a 21-element one-dimensional array. STRING$ is a 30-character string variable.

0020 DIM ARRAY(6,5), C(2,6), STRING$(28)

This statement redimensions ARRAY to a 7x6 two-dimensional array, redimensions one-dimensional array C of 21 elements to be a 3x7 (21 element) two-dimensional array, and redimensions the length of STRING$ to 28 characters.

In the following example, a program stops and arrays are redimensioned using variables with preassigned values:

STOP AT 0070

```
*
*DIM ARRAY(X,Y), C(J), STRING$(Z) )
*CON )
.
.
.
```

END

Optional program terminator.

Format

END

Why Use It?

Many BASIC systems require an END statement as the last program statement or as the terminating statement of a main program that calls one or more subroutines. We have included END in Business BASIC so that you would not have to change your END statements in order to keep your programs compatible with BASIC programs written for other systems.

What It Does

END terminates program execution. END can also stop the execution or a SWAPped-to program. If you don't use an END or STOP statement, your program terminates at the last logically executed statement.

END will not cause a message to be output to the terminal whereas a STOP message will. However, if END stops the program with unresolved GOSUBs that have not RETURNed, you will get an error message.

How To Use It

You can only use END as a program statement. It does *not* have to be the last statement in your program, but it will terminate the program when it is executed.

Example

0990 END

Licensed Material-Property of Data General Corporation

- *Statement*
- *Command*
  Function

ENTER

Bring in character formatted program source statements.

Format

ENTER { "filename" / strvar$ }

Arguments

filename a literal filename for a device or disk file containing BASIC statements or data text in character (ASCII) format.

strvar$ string variable already DIMensioned and assigned a filename value.

Why Use It?

Use ENTER to bring into your working storage any file that is in character format. Any program that you LISTed to a file is in character format and can be ENTERed. If you want to use a Business BASIC subroutine (described in Chapter 7) you must ENTER it into your program because it is a text file (with a .SL extension), not a program file. ENTER merges the new program with whatever is in working storage, replacing any statements in working storage with the same line numbers. When you want to use a Business BASIC subroutine, check the line numbers to make sure they won't replace any of your current program's line numbers, and merge the subroutine with your program using ENTER.

What It Does

ENTER searches your directory for filename; if it is not found, it will search the library directory; and if not found there, it will give you an error message. If found, ENTER will bring the new program statements or text into the working storage. When a line number of the ENTERed program matches a line number of the program currently in working storage, the ENTERed statement replaces the current program statement. If the line numbers don't match, the ENTERed statement is inserted in the proper sequence of current program statements. This is how you merge programs.

ENTER does not restore DATA statements; if you want to RESTORE you must explicitly code it. Variable assignments and file status in a partially executed program that you LISTed do *not* remain fixed when the program is ENTERed.

How To Use It

Use ENTER as a keyboard mode command or a program statement. Your filename is a file created in one of several ways -- by LISTing a program to a file, as the output of a BASIC program that used PRINT FILE statements, by the EDIT utility, or it is a file, in character format, created outside the Business BASIC system. You can use any character format input/output device (like a tape drive) to write or edit a file that you want to ENTER. See LIST in this chapter for more information about character format programs, listing files, and source files.

Examples

```
*NEW )
```

(Working storage is cleared.)

```
*ENTER "TEST1.SR" )
*ENTER "TEST2.SR" )
```

(Source programs TEST1.SR and TEST2.SR are ENTERed and merged.)

```
*LIST "FINAL.SR" )
```

(The resulting program is LISTed to disk as FINAL.SR.)

```
0900 LET NAME$="POSFL.SL"
0910 ENTER NAME$
```


Licensed Material-Property of Data General Corporation

Statement
Command
• *Function*

EOF

Check for end of file.

Format

EOF (channel)

Arguments

channel a numeric expression for the channel number of a file opened for reading (input or input/output).

Why Use It?

Use EOF to detect whether that last READ FILE or INPUT FILE bumped into the end-of-file market.

What It Does

The EOF function returns 1 if it detects an end of file; otherwise it returns 0. The EOF function cannot detect the end of a subfile, only a physical file. If the file specified by channel is not opened, EOF returns 0.

How To Use It

You frequently use EOF in an IF/THEN statement to make a conditional transfer if you're at the end of a file. IF EOF (channel) THEN statement will execute statement when the EOF returns 1. Put your EOF text after your READ FILE or INPUT FILE statement. You have to keep track of a subfile's length in order to detect the end of a subfile -- you can't use EOF. Use the status word described in MTDIO to detect the end of a tape file.

Examples

```
0010 OPEN FILE (1,3),"$PTR"
0020 READ FILE (1),A,B,C,D,E
0030 IF EOF(1) THEN 0200
0040 PRINT A,B,C,D
0050 GOTO 0020
0200 PRINT "END OF FILE"
0210 CLOSE FILE (1)
.
.
.
```

Statement
• *Command*
Function

ERASE

Erase program statements.

Format

ERASE linenum$_1$, linenum$_2$

Arguments

linenum a valid line number in your program, or a valid range of lines (i.e., linenum doesn't have to be an exact line number in the program). You need both linenum$_1$ and linenum$_2$

Why Use It?

You can use ERASE to clear a range of statements that are not needed during execution of a program, or a range of statements that would conflict with new statements being merged with a program. ERASE simplifies editing by allowing you to erase more than one line at a time.

What It Does

ERASE will remove linenum$_1$ through linenum$_2$ inclusively from your program. If no lines exist in your program in the range linenum$_1$ to linenum$_2$ you'll get an error message. If linenum$_1$ and/or linenum$_2$ do not exist but there are lines with numbers in between them, those lines are erased.

How To Use It

You can use ERASE as a program statement or a keyboard mode command. As a command, you do not have to type the keyword ERASE, just the line numbers. You must supply both line numbers. To erase one line, give that line number twice.

Examples

```
0010 ERASE 1500, 1900
```

This line erases lines 1500 through 1900.

```
*100, 100 )
```

This command erases line 100 of the program currently in working storage.

Licensed Material-Property of Data General Corporation

Statement
Command
• *Function*

ERM$

Get an error message.

Format

strvar = ERM$ { (num) | (code) }

Arguments

strvar string variable or substring to receive the error message, must be large enough to contain the error message.

num record number in the BASIC.ER file for the error message you want; error messages and their corresponding BASIC.ER record numbers are in Appendix C.

code error code for the error message you want; the code for each error message is in Appendix C, and the SYS(7) function returns this code for each error message.

Why Use It?

When you trap errors in your program (e.g., using ON ERR), you might want to have the program print an appropriate error based on the code returned by SYS(7) or based on something else. Since you can use num in ERM$ and also add new error messages to the end of the BASIC.ER file, you can generate your own unique error messages.

What It Does

ERM$ first checks to see if the num or code you supplied is negative. If you refer to Appendix C, you'll notice that the first 140 error messages have the same number for num and code. RDOS/DOS, I/O, CLI, and INFOS error messages have a negative number for code and a different, positive number for num. Thus ERM$ first checks to see if the value you supplied is negative; if it is, ERM$ knows you gave it a code; if not, ERM$ assumes you gave it a num, and finds the appropriate error message.

How To Use It

You can only use ERM$ in assignment statements (or the LET command), because you have to assign the error message to strvar. You can give ERM$ the num of the record in the BASIC.ER file that contains the error message (especially if you've written your own error messages into the BASIC.ER file), or you can give ERM$ the code of the error message, returned by the SYS(7) function. SYS(7) returns the code value of the last error or current error (if there is an error). We list the error messages supplied with Business BASIC in Appendix C in ascending order according to num (the record number in the BASIC.ER file for the error message).

Examples

```
0005 DIM ER$(60)
0010 ON ERR THEN GOTO 0800
.
.
.
0800 LET ER$=ERM$(SYS(7))
.
.
.
```

On any error in the program, control will pass to line 800, where we let ER$ equal the error message for the current error.

```
0010 DIM ER$(60)
0020 FOR I=640 TO 650
0030   LET ER$=ERM$(I)
0040   PRINT ER$
0050 NEXT I
```

This program assumes you used File Maintenance to create your own error messages for record numbers 640 to 650 in the BASIC.ER file -- see Appendix C.

Licensed Material-Property of Data General Corporation

Statement
Command
• *Function*

FILL$

Fill a string or substring with a value.

Format

strvar = FILL$(ascii *[,bytes]*)

Arguments

strvar string variable or substring you want filled.

ascii numeric expression for the decimal ASCII value of the character with which you want to fill the string.

bytes optional numeric expression for the number of bytes in strvar that you want to fill.

Why Use It?

Use FILL$ to fill a string with nulls changing its current length to its DIMensioned length. You can also FILL$ certain bytes of a string with whatever character you choose. When you pass data in a string to some routines, you must FILL$ the string with nulls to its DIMensioned length if the data does not fill the string.

What It Does

FILL$ without *bytes* fills strvar from the current position in strvar (or the beginning byte specified in a subscript to strvar) to the DIMensioned length of strvar (or the ending byte specified in a subscript to strvar) with the ASCII character whose decimal code is ascii. FILL$ with bytes fills strvar from the current position in strvar (or the beginning byte specified in a subscript to strvar) to the number of bytes you specify in *bytes*. For example, if the current position (or the beginning byte in a subscript) of strvar is 30 and you specify 10 for *bytes*, FILL$ will fill the string with ascii from position 30 to 40, unless the DIMensioned length (or ending byte in a subscript) or strvar is less than 40 -- then it only fills the string to its DIMensioned length (or ending byte).

How To Use It

You can use FILL$ only in assignment statements and the LET command, but you can concatenate several string functions and string expressions in one assignment statement. Your strvar can be a string variable, a string variable with a subscript that references a byte position to its DIMensioned length, (like A$(0) which references from the current length to the DIMensioned length of A$), and a substring whose subscripts refer to the beginning and ending byte of substring.

Examples

```
0005 REM -- FILL A$ TO END WITH BLANKS
0010 DIM A$(512)
0020 LET A$="ABCDE"
0030 PRINT LEN(A$)
0040 LET A$=A$,FILL$(32)
0050 PRINT LEN(A$)
*RUN )
5
512
*DIM B$(512) )
*LET B$=A$(1,5) )
*PRINT LEN(B$) ) 5
*LET B$(5,15)=FILL$(32,4) )
*PRINT LEN(B$) ) 9
*LET B$(9,15)=FILL$(32) )
*PRINT LEN(B$) ) 15
*LET B$(0)=FILL$(32,5) )
*PRINT LEN(B$) ) 20
*LET B$(0)=FILL$(32) )
*PRINT LEN (B$) ) 512
```


Licensed Material-Property of Data General Corporation

- *Statement*
- *Command*
  Function

FOR/NEXT

Define a program loop.

Format

```
FOR ctrlvar=expr1 TO expr2 [STEP expr3]
.
.   (Block of executable statements)
.
NEXT ctrlvar
```

Arguments

ctrlvar is a simple, nonsubscripted, numeric variable that controls the loop by being incremented or decremented. You must use the same variable in both the FOR and NEXT statements.

expr1 is the first value assigned to ctrlvar when a loop is encountered. It can be a numeric expression or a variable.

expr2 is the last value assigned to ctrlvar and the terminating value of the loop. It can be a numeric expression or a variable.

expr3 determines the amount by which you want to increment or decrement ctrlvar. If not specified, the default value is +1. A positive (+) value increments, a negative (-) value decrements.

Why Use It?

If you want to repeat the execution of one or more statements, you use a loop. FOR/NEXT statements are one way you can define a loop. The FOR statement defines how many loops to perform, and the NEXT statement marks the end of the loop. A FOR/NEXT loop can execute BASIC statements any number of times (including zero times). The loop can assign incrementing or decrementing values to variables and perform a variety of tasks. You can also execute FOR/NEXT loops repeatedly by *nesting* the loop within another FOR/NEXT loop.

What It Does

When the computer first encounters a FOR statement, the expressions -- expr1, expr2, and *expr3* -- are evaluated. If you omit STEP and *expr3*, the assumed value is +1. The ctrlvar is set to the value of expr1, and loop operation proceeds as follows:

1. The ctrlvar is tested: if *expr3* is positive (incrementing) and ctrlvar is greater than expr2, the loop immediately terminates and control passes to the statement following the corresponding NEXT statement. The ctrlvar retains the last value it had during loop execution.

   If *expr3* is negative (decrementing) and ctrlvar is less than expr2, then the loop immediately terminates and control passes to the statement following the corresponding NEXT statement. The ctrlvar retains the last value it had during loop execution.

   Otherwise, the following steps are performed:

2. Business BASIC executes the block of executable statements in the FOR/NEXT loop.

3. When the computer reaches the corresponding NEXT statement, ctrlvar is set to the value of ctrlvar + *expr3*. Then the computer returns to step 1 above and repeats the loop.

How To Use It

If you use numeric variables for your expressions, you must assign values to them before using them. Give the expressions expr1, expr2 and *expr3* positive or negative values, but note that *expr3* cannot be zero. If you want to loop zero times, just set expr2 equal to expr1. The values of expr2 and *expr3* are calculated before the loop begins, so if you use any variables in these expressions and change their values in an instruction within the loop, termination of the loop will not be affected in any way. However, if you change the value of ctrlvar with an instruction within the loop, it will definitely change the loop's termination.

Licensed Material-Property of Data General Corporation

FOR/NEXT (continued)

Comments

1. You cannot use FOR/NEXT statements as keyboard mode commands.

2. Every FOR statement must have a matching NEXT statement, and vice-versa, or you will get an error message.

3. If a loop is exited before it is completed (using GOTO or GOSUB), the FOR statement of that loop will remain active until another FOR statement with the same ctrlvar is executed. You can return to the loop if there is no other loop in the program with the same ctrlvar or, if another loop with the same ctrlvar hasn't executed yet.

4. Branching in and out of a FOR/NEXT loop is possible, but if you skip a NEXT statement and return to the loop without restarting the loop, you will get unpredictable results. If you branch out of a loop and then want to return and resume execution of the loop, you should return to the NEXT statement and execute it. If you don't, and you try to execute a FOR statement, it might be interpreted as a new nested loop within your unfinished loop, and it could cause problems.

5. You can nest FOR/NEXT loops to a depth of eight, or to a depth specified by your system manager. To properly nest a loop within another loop, make sure that the FOR and NEXT boundaries of the outer loop contain the FOR and NEXT boundaries of the inner loop. For example:

Legal Nesting

```
FOR X = ...
 FOR Y = ...
  FOR Z = ...
  NEXT Z
 NEXT Y
NEXT X
```

Illegal Nesting

```
FOR X = ...
 FOR Y = ...
 NEXT X
NEXT Y
```

Examples

```
*10 FOR J = 1 TO 10 STEP 2 )
*20   PRINT J, )
*30 NEXT J )
*40 PRINT )
*50 PRINT "J NOW HAS THE VALUE: ";J )
*RUN )
 1   3   5   7   9
J NOW HAS THE VALUE: 9
```

The ctrlvar J equals last value assigned during execution of loop, before expr2 was exceeded.

```
*10 FOR J=10 TO 2 STEP -2 )
*20   PRINT J, )
*30 NEXT J )
*40 PRINT )
*50 PRINT "LAST VALUE OF J: ";J )
*RUN )
 10   8   6   4   2
LAST VALUE OF J: 2
*

*LIST )
 0010 LET X=12
 0020 LET Y=18
 0030 FOR VALUE=X TO Y
 0040   PRINT VALUE,
 0050 NEXT VALUE
 0060 STOP
*RUN )
 12   13   14   15   16   17   18
STOP AT 0060
*
```


Licensed Material-Property of Data General Corporation

• *Statement*
Command
Function

GOSUB/RETURN

Transfer to and from a subroutine within a program.

Format

```
GOSUB linenum
.
.
.
linenum
.
.       (subroutine statements)
.
RETURN
```

Arguments

linenum is the line number entry point of a subroutine; it must exist as a program line number.

Why Use It?

A subroutine is a routine to which you can transfer control from anywhere in your program and still return to the place from which you transferred. You can write programs that use interchangeable subroutines, or make your program a series of subroutines called from a main program. It is easy and simple to read and understand programs that use subroutines to handle specific tasks.

Business BASIC also provides special prewritten subroutines to handle special functions. You can take advantage of these subroutines by using ENTER to merge them with your program (be careful, these subroutines have predefined line numbers!) and GOSUB to transfer to the pre-defined entry points of these subroutines. For more information, look up each subroutine in Chapter 7.

What It Does

GOSUB immediately transfers control to the line number specified by linenum. One by one the statements of the subroutine beginning at linenum are executed until the computer reaches the RETURN statement. The RETURN statement returns control to the statement immediately following the GOSUB statement that provided an entry point for the subroutine.

How To Use It

You can have many GOSUBs to the same subroutine, and a RETURN occurring in that subroutine would return control to the statement immediately following the specific GOSUB that you used to get to the subroutine in that instance.

You may use more than one RETURN statement in a subroutine if program logic requires the subroutine to terminate at several different places depending on a condition.

You can only enter a subroutine by using GOSUB. If you don't use a GOSUB, the RETURN statement causes an error when executed. To prevent inadvertant entry to a subroutine, you should put a STOP or GOTO statement before the subroutine so that the RETURN statement won't execute.

Subroutines may be nested to a depth of eight. Nesting occurs when a subroutine is called during the execution of another subroutine. Upon execution of the first RETURN statement, control passes to the statement immediately following the GOSUB statement last executed. The next RETURN statement causes control to pass to the next to last executed GOSUB statement, and so on.

Licensed Material-Property of Data General Corporation

GOSUB/RETURN (continued)

Examples

```
*LIST )
0010 LET A=6
0020 GOSUB 0100
0030 LET A=10
0040 GOSUB 0100
0050 STOP
0100 FOR I=1 TO A STEP 2
0110   PRINT I;
0120 NEXT I
0130 PRINT
0140 RETURN
*RUN )
 1  3  5
 1  3  5  7  9

STOP AT 0050
*
```

```
*LIST )
0010 GOSUB 0040
0020 PRINT "EXAMPLE";
0030 STOP
0040 PRINT "NEST";
0050 GOSUB 0080
0060 PRINT "TINE";
0070 RETURN
0080 PRINT "ED"
0090 GOSUB 0120
0100 PRINT "ROU";
0110 RETURN
0120 PRINT "SUB";
0130 RETURN
*RUN )
NESTED SUBROUTINE EXAMPLE

STOP AT 0030
*
```

```
*ENTER "DELREC.SL" )
*LIST 1200,1220 )
1200 LET F%=2
1210 LET R1=X3
1220 GOSUB 8600
*
```

This example shows you how to use the prewritten subroutine -- DELREC.SL. First you must ENTER "DELREC.SL" and merge it with your current program. The subroutine takes up line numbers 8600 through 8695, but 8600 is the only entry point. DELREC.SL, like all the pre-written subroutines, already has a RETURN statement -- you need not provide one. F% holds the logical file number, and R1 holds the record number.

Licensed Material-Property of Data General Corporation

• *Statement*
Command
Function

GOTO

Go to a specific statement.

Format

GOTO linenum

Arguments

linenum is an existing program statement line number.

Why Use It?

You can use GOTO anywhere in a program to transfer control to the statement specified by linenum. You can set up programming loops using GOTO statements in IF/THEN statements or in conditional statements like ON ERR.

What It Does

GOTO transfers control to the statement specified by linenum. If the statement at linenum is executable, then the statement and those following will be executed in normal sequential order. If the statement at linenum is a nonexecutable statement (like DATA or REM), then the computer executes the first executable statement following the statement at linenum.

How To Use It

Make your linenum an integer in the range -- 0000 to 9999. If linenum is not a line number in the program an error will occur. Your linenum should never be the same as the line number of the GOTO statement, or you'll be in an infinite loop (i.e., a GOTO should never "go to" itself).

Examples

```
*LIST )
0010 READ X
0020 PRINT X
0030 GOTO 0010
0040 DATA 1,2,3,4,5
0050 DATA 20,21,23
0060 STOP
*RUN )
1
2
3
4
5
20
21
23
ERROR 15 AT 0010 - END OF DATA
*
```

In this example, control will continue to transfer back to line 10 until all values for X have been read.

Licensed Material-Property of Data General Corporation

Statement
Command
● *Function*

GPOS

Current byte position in a file.

Format

GPOS (channel)

Arguments

channel numeric expression for a channel number of a file opened for sequential or random access.

Why Use It?

Use GPOS to find out what byte the file pointer is pointing to after a file access. An input or output to a file causes the file pointer to move to the byte immediately following the last byte read or written, so you use GPOS to find out where the next input or output will start.

What It Does

GPOS returns the relative byte number in the file (specified by channel) the file pointer is currently pointing. Note that the latest access to the file sets the file pointer's current position. GPOS returns a -1 if the file you specified by channel is not open. The byte position after a BLOCK READ FILE is undefined.

How To Use It

Use GPOS as a numeric expression wherever numeric expressions are permitted. A frequent use is with IF/THEN statements when you want to conditionally transfer to another routine depending on the value of the file pointer.

Examples

```
0010 OPEN FILE (0,0),"DATA"
0020 DIM REC$(50)
0030 POSITION FILE (0,0)
0040 READ FILE (0),REC$
0050 IF GPOS(0)=-1 GOTO 0600
0060 POSITION FILE (0,GPOS(0)+50)
0070 READ FILE (0),REC$
0080 IF MOD(GPOS(0),50) GOTO 0700
.
.
.
```

Line 30 positions the pointer to first byte of DATA. Line 40 reads in a record of 50 bytes. If DATA isn't opened, control passes to line 600; otherwise, we skip the next 50 bytes of DATA.

If the pointer divided by 50 produces a remainder, then the pointer is not a multiple of 50, so control passes to line 700.

Licensed Material-Property of Data General Corporation

• *Statement*
Command
Function

IF

Transfer to or execute a statement if a condition is true.

Format

IF {relexpr / expr} THEN statement

IF {relexpr / expr} {THEN / GOTO} linenum

IF {relexpr / expr} GOSUB linenum

Arguments

relexpr — two numeric or string expressions separated by a relational operator. The relational operators are: greater than (>), less than (<), equal to (=) and not equal to (<>). The expressions can also be subscripted variables.

expr — used instead of relexpr, this can be a numeric expression or variable, but not a string. If expr is 0, the condition is false; if expr <> 0, the condition is true.

statement — any legal BASIC statement except FOR, NEXT, DATA, MSG, ON ERR, ON IKEY, END, REM, and CON (you *can* use DIM, ON GOTO, ON GOSUB, and commands you use as statements).

linenum — any legal BASIC statement line number that exists in your program.

Why Use It?

Use IF statements to make decisions. If a condition is true, then execute this statement; if false, then execute the next statement after the IF statement.

What It Does

A decision concerning a condition is made based on the value of expr or the logical answer to relexpr. For example, if expr is equal to 0, the condition is false; if expr is positive or negative but not 0, the condition is true. If the relexpr A>B is true, then the condition is true; if not, then the condition is false. If the condition is true, the statement after THEN (or the GOTO or GOSUB) is executed; if false, control passes to the statement following the IF statement.

How To Use It

You only use IF as a program statement. You can put linenum or GOTO linenum after THEN: it's still a simple GOTO transfer. To make a GOSUB transfer you must use GOSUB after THEN. Otherwise, you can have most BASIC statements after THEN, including another IF statement, but you can only nest one IF within another IF. If you have a GOSUB transfer, the subroutine's RETURN statement will return control to the statement immediately following the IF.

The expressions in relexpr can be numeric or string, but you can only compare numeric to numeric and string to string, never numeric to string. If you're comparing strings or substrings using the equal (=) sign, IF will match the strings character-by-character until a difference is found, if any. If you're comparing strings using the greater than (>) or less than (<) signs, IF will compare the first characters of each string by their ASCII code value. To be equal, both strings must have the same characters, in the same order, and be of the same length. Even one extra null at the end of one of two otherwise identical strings makes that string longer.

Licensed Material-Property of Data General Corporation

Examples

```
0030 IF VALUE/100 > X*BAL THEN 0100
0040 PRINT VALUE
.
.
.
```

If the condition is true, control passes to line 100; if not, line 40 prints VALUE.

```
0010 DIM A$(20), B$(20)
0020 LET A$="ABCDEFGHIJKLMNOPQRS",FILL$(0)
0030 LET B$="ABCDEFGHIJKLMNOPQRS"
0040 IF A$ > B$ GOTO 0600
0050 PRINT "FALSE"
0055 GOTO 0065
0060 PRINT "TRUE"
*RUN )
TRUE
*
```

A$ has 19 characters, and 1 null; B$ has 19 characters. Condition is true, and control passes to statement at 0060.

```
0010 INPUT "BALANCE: $",BAL
0020 INPUT "AMOUNT OF CHECK: $,"AMT
0030 IF BAL < 100 THEN IF AMT > BAL GOTO 0600
0040 IF BAL < 100 THEN IF AMT = BAL GOTO 0700
0050 IF BAL < 0 THEN 0900
0060 IF BAL GOSUB 0500
0070 PRINT "BALANCE IS: $";BAL
0090 STOP
.
.
.
```

If BAL=0, then no transaction occurs, BAL is printed, and program stops.

Licensed Material-Property of Data General Corporation

- *Statement*
- *Command*
  Function

INPUT

Input data in character format from a terminal or from a file.

Format

INPUT [{ *FILE (channel),* / *[@ (num),] "prompt",* }] *[var [,var ...]][;]*

Arguments

channel channel number of a file if inputting from a file; file must be in character format.

@ (num) cursor positioning and terminal control expressions, which you can repeat, are described in PRINT [FILE]; they are used in conjunction with *prompt.*

prompt string literal in quotes that is output as a prompt for an input request from a terminal. This prompt replaces the question mark (?) prompt. You can enclose ASCII values in angle (< >) brackets within the string literal. This also can be repeated.

var numeric variable, string variable, or array variable, depending on the values you want to input. You can also use subscripted string variables and array variables. You must always DIMension string variables and arrays before using them. Any combination of the above is allowed if you repeat *var.* You need *var* to input data. INPUT without *var* will only display *prompt.*

Why Use It?

Use INPUT (without FILE and *channel*) to enter numeric data or strings from your terminal at runtime, and assign the data to variables. Use INPUT FILE with *channel* to enter numeric data or strings from a file that is in character format, and assign the data to variables. INPUT FILE will input actual characters from the file, not binary data, so make sure that the file is in character format. The access to files is always sequential (see "Sequential Access" in Chapter 4).

What It Does

When an INPUT statement (without FILE) is executed, you will see a question mark (?) prompt at your terminal, (unless you add a *prompt* with your INPUT statement, then *prompt* is output instead of the question mark). You respond to the prompt by typing in values. If INPUT only has one *var,* you can only type one value. If INPUT has more than one *var,* and you type only one value, it will keep asking for more values by outputting more prompts. You can supply values by answering each prompt, or you can type all the values at once by separating them with commas and terminating the whole list with a carriage return. If the values you type do not match the variable type in the INPUT statement, INPUTA outputs a backslash and question mark (\?) to signal an error and allows you to retype the value. Use INPUT USING if you *don't* want this to happen. If you use commas to separate the values, and you type too many values for that one INPUT statement, the values you supplied will be assigned to the variables, an error condition occurs, and the excess values are ignored.

INPUT FILE causes characters to be input sequentially from a character-format file. INPUT FILE always peforms sequential access even when the file is random or contiguous (see "Sequential Access" in Chapter 4).

How To Use It

You can use terminal control and cursor positioning (@) to put *prompt* anywhere on a screen (and make it blink, etc.) with the INPUT statement; but you must use terminal control with *prompt,* but use *neither* with INPUT FILE. You may separate values from a file or values typed at a terminal with commas or carriage returns, but you must terminate the entire list of values with a carriage return (<13>).

How To Use INPUT

To clarify what you should type at runtime, you might add this to your program:

```
0040 PRINT @(-40) "FOR A, B, AND C:" @(-41)
0050 INPUT A,B,C
```


Licensed Material-Property of Data General Corporation

INPUT (continued)

But you could also use INPUT statements and *prompts* in your program, and we show you two sample INPUT statements -- one with a prompt only, and one with both prompts and variables:

```
0040 INPUT @(-40), "FOR A,B, AND C:",@(-41)
0050 INPUT "TYPE A:",A,"TYPE B:",B,"TYPE C:",C
```

The @(-40) and @(-41) expressions turn a blinking field on and off. For information about terminal control and cursor positioning, see PRINT [FILE]. For the response to line 50 above, if you separate the values you type with a carriage return, each prompt is displayed; but if you answer "TYPE A:" with several values separated by commas, the prompts "TYPE B:" and "TYPE C:" are skipped, and the values you supplied are assigned to the variables. Since the INPUT statement on line 40 above did *not* end with the optional semicolon (;), line 50's INPUT request started at the next line -- i.e., a carriage return/line feed automatically occured when line 40 was executed. If you put a semicolon at the end of the INPUT statement on line 40:

```
0040 INPUT @(-40), "FOR A, B, AND C.", @(-41);
```

then line 50's INPUT request would start on the same line as line 40's, and a carriage return/line feed would *not* force it to the next line. The dimensioned length of string variables and arrays determines the size of the input values, but you can also limit the number of input characters using STMA 4,10.

How To Use INPUT FILE

As with INPUT, the type of *var* must correspond to the type of data. For example, if you're INPUTting A, B%, and C$, the first data value must be numeric, the second value must be single precision (2 bytes), and the third must be string. Commas can separate the values, but the entire list of values must terminate with a carriage return. You cannot use prompts or terminal control with cursor positioning in INPUT FILE statements.

INPUT FILE performs a sequential access, so read "Sequential Access" in the "File Access" section of Chapter 4 to find out more about sequential access. You can test for an end of file by using the EOF function, as shown in the last example.

Examples

```
0010 INPUT A,B,C
0020 PRINT A+B,B+C
*RUN )
?1,2,3 )
3  5
*

0010 INPUT "A,B,C:",A,B,C
0020 PRINT A+B,B+C
*RUN )
A,B,C: 1,2 )  ?3 )
3  5
*

0010 INPUT A,B,C;
0020 PRINT "NO RETURN"
*RUN )
?A ) \?1 ) ?2,3 ) NO RETURN
*

0010 DIM A$(5)
0020 INPUT "TYPE A$ AND NUM:",A$,NUM
0030 PRINT A$, NUM
*RUN )
TYPE A$ AND NUM: 123AB )  ?123 )
123AB  123
*

0030 DIM A$(10),B$(12),AR(3,5)
0040 OPEN FILE (1,3),"$PTR"
0050 FOR I=0 TO 3
0060   FOR J=0 TO 5
0070     INPUT FILE (1),AR(I,J),A$,B$
0080     PRINT A$,B$
0090   NEXT J
0100 NEXT I
```

This program opens the paper tape reader and inputs values to array AR. The data values on paper tape must follow a numeric-string-string-numeric-string-string-etc. sequence, or an error will occur.

```
0050 OPEN FILE (1,3),"DATA"
0060 INPUT FILE (1),A,B,C,D,E,F
0070 IF EOF(1) GOTO 1000
.
.
.
1000 PRINT "OUT OF DATA"
.
.
.
```


Licensed Material-Property of Data General Corporation

- *Statement*
- *Command*
  Function

INPUT USING

Input data in character format from a terminal or file, allowing an error trap and three alternate terminators.

Formats

INPUT USING "", *[@ (num), "prompt",]* var ... *[strvar]*

INPUT FILE (channel),USING "", var ... *[strvar]*

Arguments

var — numeric variable, string variable (already DIMensioned) or array (already DIMensioned), or subscripted string variable or array, depending on the values you want to input. Any combination of the above is allowed if you repeat var.

strvar — a special string variable at the very end of the INPUT [FILE] USING statement that will receive the value of the terminator: carriage return, form feed, or null (INPUT USING without FILE must come from terminal and always have a carriage return as the terminator). You must DIMension this string variable before using it.

channel — channel number of an opened file if inputting from a file; file must be in character format.

@ (num) — cursor positioning and terminal control expressions are described in PRINT [FILE]; they are used in conjunction with *prompt.*

prompt — string literal in quotes that is output as a prompt for an input request from the terminal (replaces the question mark (?) prompt). You can have ASCII values enclosed in angle brackets (< >) You *cannot* repeat the *prompt* argument.

Why Use It?

Use INPUT USING to input data from your terminal so that incorrect responses do not output a /? on your screen but instead cause a normal error condition that you can handle with ON ERR. Use INPUT FILE USING to input data from a file where the data coule be terminated by a carriage return, form feed, or null. INPUT FILE USING also causes a normal error that you can handle with an ON ERR if the data types are inconsistent with the variables in the INPUT FILE USING statement.

What It Does

INPUT [FILE] USING works the same way as INPUT [FILE], except for the following differences: If var is numeric and the input value can't be converted into a number, INPUT USING causes a normal error that you can trap with an ON ERR rather than outputting a /? (backslash and question mark) and waiting for input. If var is a string variable, the entire input list of values is transferred unedited (i.e., punctuation marks included) to var up to the maximum var can hold. If the first var is numeric and the next var is a string variable, then the first input value must be numeric, and the remaining input is transferred unedited (whether numeric or string) to the string variable. If *strvar* is included as the last string variable in the INPUT [FILE] USING statement, then the terminating character (carriage return, form feed, or null) will be placed in *strvar,* which can be a string expression or a string variable DIMensioned to the least 1 byte. Whenever you input from a terminal, always use a carriage return for the terminator.

How To Use It

You can use INPUT USING just like INPUT, except -- you can only have one *prompt.* Your last string variable will always be *strvar,* so if you want to input three strings, add a fourth string as the *strvar* that contains the terminator. In this situation (when inputting to three strings), you must fill the first two string variables to their DIMensioned lengths before the third string variable receives any value; but even if the third string variable receives no value, the fourth, or last, string variable, *strvar,* still receives the terminator. You can't use a semicolon at the end of INPUT [FILE] USING.

You can use INPUT FILE USING just like INPUT FILE, except -- if the first var is numeric and the first data value isn't numeric, a normal error occurs that you can trap with an ON ERR; and if your second var is a string variable, the remaining input (no matter what type) is transferred unedited to the string variable. Your last string variable STRVAR receives the terminator, which can be carriage return, form feed, or null. This differs from INPUT FILE because with INPUT FILE you must be sure that the terminator is a carriage return.

INPUT FILE USING always performs a sequential access, so read Chapter 4 to find out more about sequential access.

Licensed Material-Property of Data General Corporation

Examples

```
0010 DIM A$(132)
0020 OPEN FILE (0,0),"ASCIIDATA"
0030 INPUT FILE (0),USING "",A$,A$(LEN(A$)+1)
0040 IF EOF(0) GOTO 0300
0050 OPEN FILE (1,0), "NEWFILE"
0060 PRINT FILE (1),A$
0070 GOTO 0030
0300 CLOSE
0310 STOP
```

In this example, A$(LEN(A$)+1) is the expression for *strvar*, and it will put the terminator immediately following the rest of the data in A$.

```
0010 DIM JUNK$(100)
0020 INPUT USING "",A$
0030 PRINT A$
*RUN )
?THIS IS JUNK. "TEST",COMMA TOO! BYE<7> )
THIS IS JUNK. "TEST",COMMA TOO! BYE < 7 >
*

0010 DIM A$(50),B$(1)
0020 ON ERR THEN GOTO 0100
0030 INPUT USING "","TYPE NUMBER FIRST:",NUM,A$,B$
0040 PRINT NUM,A$,LEN(B$)
0050 STOP
0100 PRINT "ERROR!"
0110 STOP
*RUN )
TYPE NUMBER FIRST:  AB )
ERROR!

STOP AT 0110
*RUN )
TYPE NUMBER FIRST:  123 )  ?456ABC )
123   456ABC   1

STOP AT 0050
*
```


Licensed Material-Property of Data General Corporation

Statement
Command
• Function

INT

Truncate a number to make it an integer.

Format

INT(expr)

Arguments

expr numeric expression or variable you want changed to an integer.

Why Use It?

Since Business BASIC stores values as 32-bit integers, you only need the INT function when you are inputting data that is not integer.

What It Does

INT truncates to form the greatest integer not larger than expr. It does not round. When, for example, expr is 8.9999, the INT function returns 8, *not* 9.

How To Use It

Use the INT function as a numeric expression wherever you are permitted to use numeric expressions.

Examples

```
0010 OPEN FILE (0,0),"FOREIGN"
0020 INPUT FILE (0), USING "",A
0030 LET A=INT(A)
.
.
.
```

This example shows one use if INT that changes foreign data input from a file to integers when the data is in decimal notation.

• Statement
• Command
Function

KADD

Add a key to an index file (ISAM).

Format

KADD desc$, buff$, key$, recno

Arguments

desc$ index file descriptor string, either a regular or subscripted string variable DIMensioned to or with a substring length of at least 18 bytes, containing the channel, byte offset, automatic lock flag, and filename of the index file.

buff$ buffer string, either a regular or subscripted string variable DIMensioned to or with a substring length of at least 544 bytes used as a buffer to hold an index block while performing I/O to an index file.

key$ key string, either a regular or subscripted string variable, which is the key entry, and must be DIMensioned to or have a substring length of exactly the same number of bytes in the key, or the length of the key when crammed if you are using the CRM$ function.

recno numeric variable for the data record number to which you want the key to point. The data file need not exist yet.

Why Use It?

Use KADD to add key entries to an index file. In the same program that writes data to the data file, you can add keys to many index files using a KADD to each index file. If the data file already exists, but it is unsorted, you will find that it is usually faster to use KADD than IBUILD. See "Index Files" and "Creating Index Files" in Chapter 4.

Licensed Material-Property of Data General Corporation

KADD (continued)

What It Does

KADD adds key$ and recno to the index file described in desc$, using approximately a 75% blocking factor. KADD pulls index blocks into buff$ and searches them to find the proper location for the key entry so that the index is always sorted in ascending order. If you are not allowed to use duplicate keys, and KADD finds a duplicate key while searching for the proper location, the new key is not added, and recno is returned with a value of 0. If you may use duplicate keys, the add is successful; however, duplicate keys cannot have the same recno. Duplicate keys are set when you create the index file (see "Creating Index Files" in Chapter 4).

How To Use It

To use KADD, you must have a descriptor string desc$, you must DIMension buff$ to 544 bytes, and you must supply key$ and recno. You can build a descriptor string once in your program and use it for any KADD, KDEL, KFIND, or KNEXT. Your descriptor string desc$ must include the following information concatenated to form at least an 18-byte string: the physical channel number of the index file (2 bytes); the byte offset to block 0 of the index file in the physical file, or 0 if index file is not a subfile (4 bytes); a 0 when you want automatic locking of the index file to occur, or a 1 for no automatic locking (2 bytes), and the logical filename for the index file (not needed if you don't use automatic locking), padded with nulls to the end of the string (10 bytes). See "ISAM" in Chapter 4 to see what desc$ should look like and how to build it.

You must DIMension key$ to the maximum number of bytes per key entry for the particular index file. Key entries to the same index file always have the same maximum number of bytes. Keys can be alphanumeric, concatenated sub-keys, crammed strings (using the CRM$ function), or any combination of these, assigned to one string key$.

Keys should be added after you write the data record to prevent another user from trying to read the record to which the keys point before that data record is written. In multiple-user systems you should allow automatic locking of the index file and you should lock the data file using LOCK (see "File Locking" Chapter 4, and see LOCK/UNLOCK in this chapter). For more information about adding a key to an index file, see "ISAM" in Chapter 4.

Example

```
0010 DIM D$(18),B$(544),KEY$(10),REC$(128),C1(2,3)
.               :Code to fill C1 array and open files.  C1 contains
.               :file information for INDEX and DATA.  C1(1,0) is the
.               :channel number, C1(1,1) is the byte offset for INDEX.
0100 LET D$=CHR$(C1(1,0),2),CHR$(C1(1,1),4)     :Fill descriptor string.
0110 LET D$(0)=CHR$(0,2),"INDEX",FILL$(0)       :Allow index locking.
.               :Put input and checking routines here to supply KEY$
.               :and REC$ with new data.
.
0200 LET F%=2             :Logical file 2 in C1 array is DATA.
0210 LOCK 2,"DATA",0,C1(2,3)    :Lock record 0 of DATA.
0220 GOSUB 8400                 :Get available record using GETREC.SL.
0230 UNLOCK 2                   :Unlock record 0.
0240 GOSUB 9610                 :Using POSFL.SL and R1 from GETREC.SL,
                                :position to record R1, return channel in C%.
0250 WRITE FILE (C%),REC$       :Write new data record using C% channel number.
0260 KADD D$,B$,KEY$,R1         :Add new KEY$, R1 is the record pointer.
0270 IF R1<=0 THEN GOTO 0500    :If KADD is not successful (maybe a duplicate,
.                               :or R1 had the wrong value, or an error
.                               :occurred), R1 returns with a value <=0.
.
```


Licensed Material-Property of Data General Corporation

- *Statement*
- *Command*

  Function

KDEL

Delete a key from an index file (ISAM).

Format

KDEL desc$, buff$, key$, recno

Arguments

The arguments are the same as those for KADD, but note that recno only needs a value if you are using duplicate keys in the index file, and recno is returned by KDEL with the value of the record pointer if the KDEL was successful, or 0 if KDEL did not find the key.

Why Use It?

Use KDEL to delete a key entry and its record pointer from an index file. See "ISAM" in Chapter 4 for more details.

What It Does

KDEL searches for key$, and if duplicates are allowed, KDEL searches for key$ and then recno. If no match is found for key$ or for key$ and recno, recno is returned with a value of 0; otherwise, key$ and recno entries in the index are deleted and recno is returned with the record pointer.

How To Use It

To use KDEL, you must put the index file information into desc$ (see KADD), DIMension buff$ to 544 bytes, supply a value for key$, and if your index file has duplicate keys, supply a value for recno; otherwise, supply an initialized variable recno so that it can return with a value for you to check.

When you are deleting an index key for a data record, you should first try to LOCK the record to make sure nobody is using it, and to prevent anyone from trying to lock it. Then delete the index entry and the data record respectively, so that no one can try to use the key to access the record.

```
0010 DIM D3$(18),B3$(544),KEY$(10)       :D3$ is the descriptor string,
.                                        :B3$ is the buffer string.
.                                 :Set up C1, open files, etc.
.
0100 LET D3$=CHR$(C%,2),CHR$(0,4)        :C% is channel number for INDEX,
                                         :byte offset is 0.
0110 LET D3$(0)=CHR$(0,2),"INDEX",FILL$(0)      :Allow index auto lock.
0120 INPUT "KEY TO DELETE FROM INDEX: ",KEY$
0130 INPUT "DUPLICATES IN INDEX? YES(0),NO(1): ",A
0140 IF A=0 THEN GOTO 0200               :If duplicates, go get record pointer.
0150 LET R=0                             :Initialize R if no duplicates.
0160 KDEL D3$,B3$,KEY$,R                 :Delete key and pointer, return R.
0170 IF R=0 THEN GOTO 0400               :If key not found, go to 400.
0200 REM -- IF DUPLICATES EXIST
0210 INPUT "KEY POINTS TO WHAT RECORD IN DATA FILE? ",R
0220 GOTO 0160                           :Go back to KDEL.
.
.
.
```


Licensed Material-Property of Data General Corporation

- *Statement*
- *Command*
  Function

KFIND

Find a key in an index file (ISAM).

Format

KFIND desc$, buff$, key$, recno

Arguments

The arguments are the same as for KADD, except that buff$ must be unique to this index file, and recno is an initialized numeric variable that receives the record pointer.

Why Use It?

Use KFIND to find any key in an index file, or to find the first key in an index file. You can then use KNEXT to read the index file sequentially from the key found by KFIND. See "ISAM" in Chapter 4.

What It Does

KFIND searches the index file described by desc$ by pulling blocks into buff$ (a subsequent KNEXT requires that you use the same buff$ and not destroy it). If KFIND finds an exact match for key$, recno returns with the record pointer associated with that key. If KFIND can't find an exact match, it finds a key with a value slightly greater than key$ (approximate match), and returns recno with the negative of the record pointer associated with the found key. KFIND searches to the end of the index for key$ or a value slighty greater; if it doesn't find key$ it will find the highest value in the index which is the end-of-index value, and return recno with a zero value.

How To Use It

You must build a desc$ string, DIMension buff$ and supply a value for key$. After using KFIND, you should check recno -- if recno is negative, an approximate match was found, and you should check it and perhaps use KNEXT to get the next key in sequence; if recno is 0, nothing like key$ was found in the index; if recno is a positive record pointer, then you found the key for which yu were looking.

Example

```
0010 DIM DI2$(18),BI2$(544),KEYI2$(12)
.                        :Build descriptor string DI2$, and open
.                        :index and data files.
.
0090 LET R=0             :Initialize record pointer for KFIND to return.
0100 INPUT "KEY: ",KEYI2$
0110 KFIND DI2$,BI2$,KEYI2$,R    :Go find key in index, return R.
0120 IF R=0 THEN GOTO 0800       :If nothing is found, go to 800.
0130 IF R<0 THEN GOTO 0400       :If approximate match, go to KNEXT routine
                                 :at 400.
0140 LET R=R1                    :Found it, supply R1 for POSFL.SL.
0150 GOSUB 9610                  :Position to data record using POSFL.SL.
.
.
.
```


Licensed Material-Property of Data General Corporation

- *Statement*
- *Command*
  Function

KNEXT

Find the next key in sequence in index file after KFIND (ISAM).

Format

KNEXT desc$, buff$, key$, recno

Arguments

The arguments are the same as for KADD, except that buff$ must be unique to this index file, and key$ (DIMensioned to the key length) will receive the key and recno will receive the key's associated record pointer.

Why Use It?

Use KNEXT after a KFIND to find the next key in sequence, and use subsequent KNEXTs (or a KNEXT loop) to read the index sequentially from that point. A frequent use of KNEXT is -- to find duplicate keys or approximate matches; another is to process a data file in a sorted order by reading an index sequentially from the beginning.

What It Does

KNEXT will return in key$ the next key in sequence after a KFIND and the key's associated record pointer in recno from the index file described in desc$. KNEXT pulls an index block into buff$ and uses buff$ again when KNEXT is executed again, so buff$ must not be disturbed between each KNEXT. When KNEXT reaches the last index entry, a subsequent KNEXT will return recno with a zero value, and any additional KNEXTs cause an end-of-file error.

How To Use It

To use KNEXT, you must first use KFIND. You must have desc$, buff$, and key$ DIMensioned to the maximum length of your keys in the index, and recno initialized to some value. KNEXT will return key$ and recno, but you should check recno to see if you reached the end of the index. KNEXT needs to have a separate buff$ for each index if you are accessing more than one index in one program, because it needs to preserve the information in buff$ for the next KNEXT to the same index.

Example

```
0010 DIM DI2$(18),BI2$(544),KEYI2$(12)
.                        :build descriptor string DI2$, and open
.                        :index and data files.
.
0090 LET R=0             :Initialize record pointer for KFIND to return.
0100 INPUT "KEY: ",KEYI2$
0110 KFIND DI2$,BI2$,KEYI2$,R    :Go find key in index, return R.
0120 IF R=0 THEN GOTO 0800       :If nothing is found, go to 800.
0130 IF R<0 THEN GOTO 0400       :If approximate match, go to KNEXT routine
.
.
.
0400 KNEXT DI2$,BI2$,KEYI2$,R    :Find next one, put its pointer in R.
0410 IF R=0 THEN GOTO 0800       :If at end-of-index, go to 800.
0420 PRINT KEYI2$                :Let's look at it.
0430 INPUT "IS THIS WHAT YOU WANT? YES(0),NO(1):",A
0440 IF A THEN GOTO 0400         :If no, go back and get the next key.
0450 PRINT "OK, YOU GOT IT"
0460 LET F%=3                    :Data file is logical file 3 in C1 array.
0470 LET R=R1                    :Give R1 the record pointer found with KEYI2$.
0480 GOSUB 9610                  :Position to record R1 in data file.
.
.
.
```


Licensed Material-Property of Data General Corporation

LEN

Find the current length of a string.

Format

LEN(strvar)

Arguments

strvar string variable or substring of which you want to find the current length.

Why Use It?

Although a string has a maximum DIMensioned length, its current length is the number of characters currently in the string, and this could vary a great deal. The LEN function could also come in handy when you use it with the TRUN$ function to determine exactly h w much data you received on input from a file, regardless of the DIMensioned length of the string that received the data (since file input pads the receiving string with nulls).

What It Does

LEN returns the value of the number of characters currently in strvar including spaces, punctuation, special characters, and nulls. If there is absolutely nothing in the string (including no nulls), the LEN function returns a 0.

How To Use It

Use LEN as a numeric expression wherever numeric expressions are allowed. If a string is padded with nulls or spaces (after the data), you can use the TRUN$ function (see TRUN$) and then the LEN function to find out where the data stops in the string.

Examples

```
0005 DIM A$(100)
0010 INPUT "TYPE 4 CHARACTERS: ",A$
0020 IF LEN(A$)=4 THEN GOTO 0040
0025 PRINT "YOU BLEW IT, TRY AGAIN."
0030 GOTO 0010
.
.
.

*DIM A$(512) )
*PRINT LEN(A$) ) 0

*LET A$="" )
*PRINT LEN(A$) ) 0

*LET A$=12345 )
*PRINT A$ ) 12345
*PRINT LEN(A$) ) 5

*LET A$=FILL$(0,30) )
*PRINT LEN(A$) ) 30
*LET A$=FILL$(0) )
*PRINT LEN(A$) ) 512
```


Licensed Material-Property of Data General Corporation

- *Statement*
- *Command*
  Function

LET

Assign a value to a variable.

Format

[LET] variable=expression

Arguments

variable — a numeric or string variable that can be subscripted. If numeric, this initializes the variable. If string, the string variable must have been DIMensioned. If variable ends with the triple precision (#) sign, a 6-byte assigment will occur (see Chapter 2).

expression — a numeric or string expression. If variable is numeric, expression must be numeric. Numeric expressions can have subscripted numeric variables, numeric functions, user-defined functions, and constants. If variable is a string, expression can have string functions, string literals in quotes, and even numeric expressions without quotes in order to make a string from a number.

Why Use It?

You need LET to assign values to variables, to initialize variables (INPUT and READ also initialize variables), to assign values to array elements, to make a string of a number, to do simple or complex calculations, etc.

What It Does

LET will create a numeric variable for you, and assign a value to it. You can even create an 11-element one-dimensional array or a 121-element two-dimensional array with LET statements without having to DIMension the arrays. You *can't,* however, assign a string value to a string variable without DIMensioning the string variable first. All assignments are 2-word (4-byte) allocations, unless you use the "triple precision" 3-word (6-byte) option available during system generation.

How To Use It

You can use LET as a keyboard command to perform quick calculations or assignments when debugging a program. The keyword LET is optional, but when you LIST the program with LET statements, the keyword LET will show up even if you didn't type it.

Numeric expressions can have constants, variables, subscripted variables, numeric functions, user-defined functions, and arithmetic symbols. For more information about expressions, arithmetic priorities, and symbols, see Chapter 2.

String expressions can have string literals in quotes, string variables, subscripted string variables, string functions, and string operators, as described in Chapter 2.

You can assign a numeric expression to a string variable without using quotes (e.g., LET A$=1234, LET B$=3+4: A$ will contain the string "1234" and B$ will contain the string "7"). The assignment creates a string of digits that you can convert back to a number using the VAL.SL subroutine (see VAL.SL in Chapter 7).

Variables are always assigned from right to left; i.e., your variable is always on the left of the equal sign (=), and the value you want to assign to it is always on the right.

Examples

```
*X=2 )
*Y=1495 )
*LET X=X*Y )
```

When using LET as a command, LET is optional.

```
0010 DIM A$(10),B$(20),C$(20)
0020 LET A$="ABC"
0030 PRINT LEN(A$)
0040 LET B$="ABC",A$,FILL$(0)
0050 PRINT B$
0060 PRINT LEN(B$)
0070 LET C$=542
0080 LET VAL=678
0090 LET C$=C$,VAL
.
.
.
```

The length of A$ is 3, not 10. B$ contains "ABCABC" and is padded with nulls to be 20 characters long. C$ contains "542" on line 70, and "542678" on line 90.

```
*LET A#=B#+C )
```

A# and B# are triple precision variables that can hold 6 bytes (3 words) of binary data.

Licensed Material-Property of Data General Corporation

LIST

List the character contents of a program to a file or a terminal.

Format

LIST [{ *linenum* / *TO linenum*$_2$ / *linenum*$_1$ {,/*TO*} *linenum*$_2$ }] [*"filename"*]

Arguments

linenum line number when you only want one line listed.

linenum$_1$ line number of first line in a range of lines you want listed. If it doesn't exist, range will begin at the next higher line number.

linenum$_2$ line number of last line in a range of lines you want listed. If it doesn't exist, range will terminate at the lowest line closest to it.

filename a literal filename of a disk file or device, expressed as a string literal in quotes. If it already exists, you get an error message.

Why Use It?

Use LIST without *filename* to list the contents of your program in working storage on the terminal. You can LIST all the lines, a range of lines, or one line. When you only LIST one line, it is automatically put in the edit buffer so that you can use keyboard editing commands on it (see Chapter 2 for the keyboard editing commands). When you LIST a range of lines or all the lines, the last line LISTed remains in the edit buffer for you to edit. LIST with *filename* lists all or part of your program in working storage to a file named by *filename*. The file can be a device like the line printer, or a disk file to hold ASCII characters (e.g., a file with which you can use the EDIT utility).

What It Does

LIST will output all of a program or the lines specified to a disk file or device specified by *filename*. If you do not specify a *filename* LIST outputs the program or the lines specified to your terminal. If a program is LISTed to a disk file, *filename* is put in your directory (it must not already exist) and you can retrieve it by using *filename* with the ENTER command (see ENTER, this chapter). If *filename* is a device, the output would go to the device in character format.

How To Use It

To output part or all of a program to a disk file, specify *filename* of the new file. To output part or all of the current program to the line printer, specify "$LPT" as your *filename* (for other device names, see Chapter 1). To specify part or all of the current program in working storage, use these formats:

LIST

output the entire program starting at the lowest numbered line.

LIST *linenum*

output only the line *linenum*.

LIST TO *linenum*$_2$

output all lines from the lowest line number to and including *linenum*$_2$.

LIST *linenum*$_1$ *TO linenum*$_2$ or LIST *linenum*$_1$, *linenum*$_2$

output all lines starting at *linenum*$_1$ through and including *linenum*$_2$. *Linenum*$_2$ *must be greater than linenum*$_1$.

Licensed Material-Property of Data General Corporation

LIST (continued)

Examples

```
*LIST )
0010 LET X=10
0020 INPUT BAL
0030 INPUT PAY
0040 LET BAL=BAL-PAY
0050 PRINT "BALANCE: ",BAL
0060 LET X=X-1
0070 IF X THEN GOTO 0020
0080 STOP

*LIST 20,40 )
0020 INPUT BAL
0030 INPUT PAY
0040 LET BAL=BAL-PAY

*LIST TO 30 )
0010 LET X=10
0020 INPUT BAL
0030 INPUT PAY

*LIST 10 )
0010 LET X=10
*.C/X=10/X=20 )
0010 LET X=20
```

".C" is a keyboard editing command, see Chapter 2.

```
*LIST TO 30 )
0010 LET X=20
0020 INPUT BAL
0030 INPUT PAY

*LIST "$LPT" )
```

Output the whole program to the line printer.

```
*LIST "BAL.LS" )
```

Output to a listing file BAL.LS.

LOAD

Retrieve a SAVEd program.

Format

```
LOAD "filename"
```

Arguments

filename a literal filename in quotes of a binary core image file created by a previous SAVE or REPLACE.

Why Use It?

Use LOAD to retrieve a program you previously SAVEd or created using REPLACE. LOADing a SAVEd program is quicker than ENTERing a LISTed program. After you have LOADed a program, you can LIST it, edit it, REPLACE it, or RUN it. RUN filename will do an automatic LOAD (see RUN in this Chapter).

What It Does

LOAD clears the current program and data from working storage (like a NEW command), and retrieves the program named by filename. The new program becomes the current program in your working storage. If filename is a disk file, LOAD first searches your directory for it, then the library directory; if not found, you get an error message. The file may be on disk or may be on a binary input device like a punch card reader. Do not confuse this LOAD with the BASIC CLI command LOAD used to restore files from a mag tape backup (see LOAD in Chapter 8).

How To Use It

LOAD can only be used as a keyboard mode command (do not execute LOAD from the BASIC CLI -- there is another LOAD for the BASIC CLI, see Chapter 8). You must specify filename. You can only LOAD programs that were previously created by a SAVE or REPLACE, or Business BASIC utility programs that are already SAVEd. RUN filename and CHAIN filename will do automatic LOADs as part of their execution.

Examples

```
*LOAD "DUMP" )
```

a program named DUMP is LOADed into working storage.

Licensed Material-Property of Data General Corporation

- *Statement*
- *Command*
  Function

LOCK/UNLOCK

File lock/unlock device.

Format

LOCK iden, { "filename" / strvar } , start, recsize *[,time]*

UNLOCK *[iden]*

Arguments

iden — numeric expression for the lock identifier, used to identify a specific lock in a program.

filename — string literal in quotes that is the logical or physical filename of file you want locked. If logical, every user must use logical; if physical, every user must use physical; if some other convention, every user must use that convention.

strvar — regular or subscripted string variable representing filename.

start — numeric expression for the logical or physical starting byte number of the lock area: the convention you use (logical or physical) must be consistent with every other user.

recsize — numeric expression for the size of the record in the lock area; i.e., the record you want to lock.

time — numeric variable (optional) containing the amount of time to wait for an UNLOCK when a LOCK suspends; it is returned with a value of 57 if the LOCK execution suspends, or a value of 56 if you try to use two LOCK statements with the same iden without an intervening UNLOCK.

Why Use It?

Use LOCK/UNLOCK to resolve conflicts between programs which share the same open file. (You don't need locks if you open your files as "exclusive".) Use LOCK to lock a record in a file so that no one else can LOCK the same area, and use UNLOCK to free the locked area so that someone else can LOCK it.

What It Does

LOCK/UNLOCK is essentially a communications device -- it doesn't actually prevent anyone from accessing the file area. LOCK prevents anyone else from using a LOCK on the ame file area, so if all users follow the convention of trying to LOCK a record before accessing it, LOCKs will prevent another user from using a LOCK on the same file area, and it will thereby prevent the user from accessing the record. See "File Locking" in Chapter 4.

LOCK does not lock only a record; LOCK locks the one or more blocks (512-byte sectors) that contain the record. LOCK divides start by 512 to get the starting block number for the lock area, and divides (start + recsize) by 512 to get the ending block number of the lock area. LOCK then examines the lock table (a list of locks in the entire system) to see if any other job LOCKed that filename (the first ten characters of filename are compared to filenames in the lock table). If another job also locked filename, the starting and ending block numbers of the lock area you supplied are compared with the starting and ending block numbers of the lock area already locked. If there is an overlap, your LOCK is not successful. Your job is suspended for the amount of time in *time*, your job is suspended until the other user UNLOCKs that filename, or your job is suspended until an interrupt occurs. In all cases, if your LOCK is suspended, the *time* variable returns with a value of 57. If there is no overlap, or if filename doesn't exist in the lock table, your LOCK is successful and your filename with your lock area is recorded in the lock table to prevent another user from locking that file area.

An UNLOCK deletes your filename and lock area from the lock table so that someone else can lock it. UNLOCK with *iden* will refer only to the LOCK with the same iden; UNLOCK without *iden* (or CLOSE, NEW, or BYE) unlocks all locked files in your program.

Lock identifiers (iden) are unique to each job; i.e., you can have from 1 to 32767 idens that are unique to your terminal -- another terminal's iden of the same value does not refer to the iden of your LOCK/UNLOCK. You can find out what locks you have set at your terminal by running the LOCKS utility program (see Chapter 7) which lists all your locks in the sequence of your idens. If you try to perform two LOCK statements at your terminal with the same iden, without an intervening UNLOCK, error 56 occurs. If you supplied a *time* variable, *time* is returned with a value of 56.

Licensed Material-Property of Data General Corporation

LOCK/UNLOCK (continued)

How To Use It

A LOCK only prevents another user from locking the same file area, so to control file access using file locking, everyone must agree to the same lock conventions, and everyone must agree to attempt to LOCK a record before accessing the record. We suggest that you use logical filenames instead of physical filenames. Otherwise, if someone tried to LOCK a file area in a physical file, and someone else tried to lock the same file area using the subfile (logical) filename instead of the physical filename, both locks would be successful, though you'd only want one of them to be successful. You also have to decide which convention to use for start-- logical starting byte within a logical file (could be a subfile), or physical starting byte in physical file whether or not subfiles exist; otherwise, the same area might be represented differently and locked by different users, which defeats the whole purpose of LOCK.

Use *time* to specify how many tenths of a second a LOCK should suspend while waiting for an UNLOCK or interrupt. When the amount of time in *time* has elapsed during a suspend, control is returned to your program, and you should then check *time* to see if it has a value of 57 (meaning an UNLOCK has not yet occurred to unlock the file area you want to lock). Use *time* to avoid deadly embraces that occur when you want to lock an area that is already locked by a user, and the user wants to lock an area you already locked, and neither of you can perform an UNLOCK while your jobs are suspended.

See "File Locking" in Chapter 4 for more information.

Example

```
.               :Set up C1 array with info about subfile SUB1 and
.               :other files.  SUB1 is logical file 2 in C1; note
.               :C1 elements as arguments to LOCK.  Open files here.
0100 LET T=0            :Initialize T, do not give it a time.
0110 LOCK 1,"SUB1",0,C1(2,3),T          :Lock record 0 of SUB1, no time delay.
0120 IF T=57 THEN GOTO 0110             :If suspended (area already locked),
                                        :keep trying LOCK.
0130 LET F%=2                           :SUB1 is logical file 2.
0140 GOSUB 8400                         :Get available record (GETREC.SL).
0150 UNLOCK 1                           :Unlock record 0, GETREC.SL is done.
0160 GOSUB 9610                 :Using R1 from GETREC.SL, use POSFL.SL to
                                :position to record R1.
0170 LET T=30                   :Set time to 3 seconds.
0180 LOCK 2,"SUB1",R1*C1(2,3),C1(2,3),T :R1 record number * record size in C1
                                        :equals start of lock area.
0190 IF T=57 THEN GOTO 0500             :If lock is unsuccessful, go to 500.
.                                       :If lock is successful, write the
.                                       :new record (code goes here).
.
0250 UNLOCK                             :Unlock any and all locks when done.
0260 CLOSE                              :Close all open files.
0270 STOP
.
.
.
0500 REM -- ROUTINE TO TRY LOCK AGAIN
0510 PRINT "LOCK AREA ALREADY LOCKED."
0520 INPUT "DO YOU WANT TO TRY AGAIN? YES(0),NO(1): ",A
0530 IF A THEN GOTO 0560
0540 PRINT "WE'LL TRY AGAIN-- YOU CAN ALWAYS USE IKEY TO GET OUT";
0550 PRINT "OF A LOCK SUSPEND."
0555 GOTO 0180                  :Go back, try LOCK again.
0560 UNLOCK 2                   :If "no" above, unlock and move on.
.
.
.
```


Licensed Material-Property of Data General Corporation

Statement
Command
• *Function*

MAX

Find the maximum of two expressions.

Format

MAX ($expr_1$, $expr_2$)

Arguments

$expr_1$ and $expr_2$ two numeric expressions or variables that you want to compare.

Why Use It?

At some point in your program you might want to choose the larger of two unknown values; you might want to set up a conditional transfer statement that executes one routine if $expr_1$ is larger, and another routine if $expr_2$ is larger.

What It Does

MAX picks the larger of $expr_1$ and $expr_2$ and returns it. If $expr_1$ and $expr_2$ are equal, MAX returns their value.

How To Use It

Use MAX as a numeric expression wherever numeric expressions are allowed.

Examples

```
0010 INPUT X, Y
0020 PRINT MAX(X, Y)
```

```
0010 DIM X$(100), Y$(100)
0020 INPUT X$, Y$
0030 PRINT MAX(LEN(X$), LEN(Y$))
```

Statement
Command
• *Function*

MIN

Find the minimum of two expressions.

Format

MIN ($expr_1$, $expr_2$)

Arguments

$expr_1$ and $expr_2$ two numeric expressions or variables that you want to compare.

Why Use It?

At some point in your program you might want the smaller of two unknown values; or you might want to set up a conditional transfer statement that executes one routine if $expr_1$ is smaller, and another routine if $expr_2$ is smaller.

What It Does

MIN picks the smaller of $expr_1$ and $expr_2$ and returns it. If $expr_1$ and $expr_2$ are equal, MIN returns their value.

How To Use It

Use MIN as a numeric expression wherever numeric expressions are allowed.

Example

```
0010 INPUT X, Y
0020 IF MOD(X, Y) = X THEN GO TO 0100
.
.
0010 IF X = Y THEN GO TO 0010
.
.
```


Licensed Material-Property of Data General Corporation

Statement
Command
• *Function*

MOD

Find the remainder after dividing two expressions.

Format

MOD(expr$_1$, expr$_2$)

Arguments

expr$_1$ a numeric expression or variable; the dividend.

expr$_2$ a numeric expression or variable; the divisor.

Why Use It?

Use MOD when you want to determine the remainder after dividing two numbers. A frequent use of MOD is to determine whether expr$_1$ is a multiple of expr$_2$ or whether expr$_2$ is a factor of expr$_1$.

What It Does

MOD divides expr$_1$ and expr$_2$ using this equation:

remainder = expr$_1$ - INT(expr$_1$ /expr$_2$) * expr$_2$

For example, MOD(17,5) equals 2. If the remainder is 0, then expr$_1$ is a multiple of expr$_2$ (and expr$_2$ is a factor of expr$_1$).

How To Use It

Use MOD as a numeric expression wherever numeric expressions are allowed.

Examples

```
0010 INPUT X
0020 LET Y=10
0030 PRINT "REMAINDER AFTER DIVIDING X BY 10:"
0040 PRINT MOD(X,Y)

.
.
0090 POSITION FILE (0,R*RSIZE)
0100 READ FILE (0),REC$
0110 IF MOD(GPOS(0),RSIZE) <> 0 GOTO 0500
.
.
.
```

If the pointer (GPOS) is not a multiple of the record size (RSIZE), control passes to line 500.

Licensed Material-Property of Data General Corporation

MSG

Send a message to another terminal.

Format

MSG, *[terminal,]* message

Arguments

terminal port number of the terminal to receive message. It can be a number, numeric expression, or variable you already DIMensioned and assigned a value. You can find out which terminals are logged on and what their numbers are by running the utility program PORTS or using the BASIC CLI command STAT. If you do not specify *terminal,* message will be sent to anyone with an OP (system operator) account.

message text of message you want to send, expressed as either a string literal in quotes or a predefined string variable.

Why Use It?

Use MSG to send a message to another terminal or to the system operator.

What It Does

When you send a message to a terminal or to the operator, the receiving terminal prints your account specifier (set by HELLO at log on -- your answer to "ACCOUNT:"), your terminal number, and message. If the receiving user set the no message bit using STMA 6,4 (see STMA in this chapter), you will get an error message; the message you sent will not be received.

How To Use It

Use MSG as a keyboard command only. First, find out which terminals are logged on and their numbers by using the BASIC CLI command STAT (see STAT in Chapter 8) or by running the utility program PORTS (see PORTS in Chapter 7). If you want to send a message to anyone with the OP (system operator) account, do not specify *terminal.* Your message can be a string literal in quotes or a previously DIMensioned and assigned string variable.

Examples

```
*MSG,2,"WHAT ABOUT LUNCH?" )
```

At terminal 2, this message would appear:

FROM AATLB6:4 WHAT ABOUT LUNCH?

AATLB6 is the account specifier for the message-sender, 4 is AATLB6's terminal number. AATLB is the account that signed on at terminal 4; the appended "6" means that terminal 4 is a type 6 terminal.

```
*MSG,4,"YOUR TREAT? OK!" )
```

At terminal 4, the reply would come back from account ABSCR:

FROM ABSCR6:2 YOUR TREAT? OK!

```
*MSG, "LET'S GO EVERYBODY!" )
```

This message will be received by any and all system operators with the OP account. Presumably the system operator(s) will gather everyone for lunch.

Licensed Material-Property of Data General Corporation

- *Statement*
- *Command*

  Function

MTDIO

Direct I/O control of magnetic tape and cassette.

Formats

Format 1:

```
MTDIO channel, func, strvar, er
```

Format 2:

```
MTDIO channel, func, records, er
```

Format 3:

```
MTDIO channel, func, er
```

Arguments

channel numeric expression for channel number of a file opened in mode 7 (mag tape I/O).

strvar string variable or subscripted string variable that contains data bound for output to tape, or receives data input from tape. You must have already DIMensioned strvar.

records numeric expression for the number of records you want to be skipped; it must be in the range from 0 to 4095. (If 0, then 4096 records are skipped, otherwise the value you specified equals the number of records skipped.)

er numeric variable that receives the tape status or error code as bit flags in the right-most 2 bytes of the variable.

func code number for one of the following functions:

Code	Function
1	read a tape record into strvar (format 1).
2	rewind the tape to the beginning (format 3).
3	not used.
4	skip records number of records in tape, moving forward (format 2).
5	skip records number of records in tape, moving backward (format 2).
6	write a tape record from strvar (format 1).
7	write an end-of-file mark on the tape (format 3).
8	read the tape drive status into er (format 3).

Default parity is odd. Make your func code negative if the tape expects even parity, but ask your system manager first.

Why Use It?

Use MTDIO to read or write magnetic tapes or cassettes. You can read data, write data, skip records, and rewind the tape using MTDIO statements in your program or MTDIO commands at your terminal. MTDIO can handle any tape format, since it reads in records whose size you determine by DIMensioning strvar. You can have a record size in the range of 2 to 8192 bytes.

Licensed Material-Property of Data General Corporation

MTDIO (continued)

What It Does

When MTDIO is reading from a tape if you have DIMensioned strvar to be larger than the amount of bytes actually read, strvar is truncated to the actual number of bytes found; if you have DIMensioned strvar to be less than the amount of bytes in the tape record read, strvar is filled to capacity and you lose the excess bytes (er does not indicate this). RDOS tape format requires an even number of bytes bacause all RDOS/DOS tape transfers are word (2-byte) transfers. If your tape is not in RDOS/DOS format, you may get byte transfers and an odd last byte -- this condition is indicated in er.

When writing a tape record, the size of the record equals the DIMensioned size of strvar or the specified length of strvar if you subscripted strvar (i.e., a substring), regardless of the current length of strvar. When skipping tape records, a value of 0 for records causes the maximum of 4096 records to be skipped either forward of backward; otherwise, a value between and including 1 to 4095 for records will cause the specified number of records to be skipped, forward or backward. If the number of records you want to skip is greater than the number remaining in the file (moving forward), the tape stops when it hits the end-of-file marker. You can test for an end of file by checking the status word returned in er.

After an MTDIO execution, er contains the status word for the tape drive unless an RDOS/DOS error occurs. An error causes the appropriate error code to be returned in er. End of file, end of tape, parity, and other tape drive errors can only be determined by checking bits of the status word. These errors are not returned as BASIC errors. The status values are shown in Figure 3-3, and it shows what each bit value signifies.

VALUE	BIT POSITION																SIGNIFIES
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
32768	1	•	0	•	0	•	•	•	•	0	•	0	0	0	•	0	Error. (Bits 14, 12, 10, 9, 8, 7, 5, or 1, are the error bits, at least one is set).
16384	•	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Data late (also sets bit 15).
8192	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	Tape rewinding
4096	•	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	Illegal command (also sets bit 15).
2048	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	High density (always set for cassettes).
1024	•	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	Parity error (also sets bit 15).
512	•	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	End of tape (also sets bit 15).
256	•	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	End of file. (also sets bit 15).
128	•	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	Beginning of tape (also sets bit 15).
64	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	Nine track tape (always set for cassettes).
32	•	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	Bad tape (also sets bit 15).
16	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	Send clock.
8	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	First character.
4	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	Write protected. Can't be written to.
2	•	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	Odd character (odd byte) that also sets bit 15.
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	UNIT READY.

2 BYTES = **er**

A "1" in a bit signifies that the bit is set, and "Value" is the value you use (with AND) to check if that one and only bit is set. Normally you'd only check one bit at a time, but if you wanted to check more than one bit, add the values together and use the total value to AND with **er.** A "•" in a bit signifies that if the bit in this word is set (signified by a "1"), at least one of the "•" bits will also be set (if only one "•" bit, it will be set).

SD-01177

Figure 3-3. Status Values

Licensed Material-Property of Data General Corporation

MTDIO (continued)

How To Use It

To use MTDIO, you must first open the desired tape device in mode 7 on channel. Device names are described in Chapter 4.

When reading a tape with MTDIO, your strvar should be of sufficient size to contain the largest expected record, since you lose any bytes that don't fit. You must request an even number of bytes. If you know your tape ends on an odd byte, you should read in the number of bytes plus one to make it even, and then discard the last byte (it will be a duplicate of the one before it). You must determine any error or tape drive status condition (like end of file, end of tape, parity error, etc.) by checking bits of the status word (see Figure 3-2). The variable er receives the status word, so use the AND function to AND er with the value of the bit flag you want to check. If you want to check more than one bit flag at once, add the values of the bit flags together and use the total value in the AND function with er. If you are reading a tape whose records were written by the RDOS/DOS CLI XFER or BASIC CLI XFER command (XFER writes 512-byte blocks), you must DIMension strvar to 512 bytes (or substring strvar to at least 512 bytes), for each MTDIO read. When reading into a substring, the substring must begin on an odd byte (1,3,5...) and be an even number of bytes long.

When you write a tape record from strvar, strvar must be an even number of bytes in length. If strvar is a substring, it must start on an odd byte (1,3,5...), and be an even number of bytes long. You should remember to check for an end-of-tape condition in the status word in er when writing long tape files; otherwise, the tape may run off the reel. After you've written all your records to the tape, you should write an end-of-file mark using func 7 to prevent attempts to read beyond the end of file when the tape is used later. RDOS/DOS expects two end-of-file marks after the last file on a tape.

When skipping either forward or backward to records on a tape, specify records as a number in the range from 1 to 4095. If you specify 0, 4096 (the maximum) records will be skipped. Use 0 in records with a forward skip to get to the end of the file quickly (it will stop at the end-of-file marker), and use 0 in records with a backward skip to go back to the exact beginning of the tape file (it will stop at the beginning-of-tape marker).

When rewinding a tape, use a loop to check the status word in er to see if the tape is still rewinding or if the unit is ready. The execution of MTDIO is fast, but tape drives are not as fast, and you don't want to introduce more errors by trying to read or write a tape that is still rewinding.

Licensed Material-Property of Data General Corporation

Examples

```
0010 DIM REC$(80)                  :Tape records are 80 bytes long.
0020 OPEN FILE (1,7),"MT0:0"       :Tape device MT0, file 0.
0025 LET ER%=0                     :Initialize ER% to receive status.
0030 MTDIO 1,1,REC$,ER%            :Read tape record into REC$, and tape status
                                   :into ER%.
0040 IF AND(ER%,256) THEN GOTO 0300     :Test for an end-of-file.
.                        :Process record, go read next record, etc.
.
.
0300 MTDIO 1,5,0,ER%               :Back up 4096 records or to beginning of file.
0310 INPUT "REC NUMBER OF NEW REC: ",NUM
0320 MTDIO 1,4,NUM-1,ER%           :Go forward to record NUM.
0330 INPUT "TYPE NEW RECORD: ",REC$
0340 MTDIO 1,6,REC$,ER%            :Write new record.
.
.
.
0500 REM -- WRITE NEW LAST RECORD AND NEW EOF MARK
0510 INPUT "TYPE LAST NEW RECORD: ",REC$
0520 MTDIO 1,4,0,ER%               :Go forward to the end-of-file.
0530 MTDIO 1,6,REC$,ER%            :Overwrite end-of-file mark with new record.
0540 MTDIO 1,8,ER%                 :Read tape drive status only into ER%
0550 IF AND(ER%,32767) THEN GOTO 1000   :If error, go to line 1000.
0560 MTDIO 1,7,ER%                 :Write an end-of-file mark on the tape.
0570 MTDIO 1,7,ER%                 :Write another end-of-file mark.
0580 MTDIO 1,2,ER%                 :Rewind the tape.
0590 IF AND(ER%,8192) THEN GOTO 0620    :Set up loop to check if tape is
                                        :still rewinding.
0600 CLOSE FILE (1)                :Tape rewound, close the file.
0610 STOP
0620 GOTO 0590                     :If tape is still rewinding, go and recheck.
.                                  :This loop will continue until tape is rewound.
.
```


Licensed Material-Property of Data General Corporation

- *Statement*
- *Command*
- *Function*

NEW

Clear your working storage.

Format

NEW

Why Use It?

Use NEW to clear your working storage so that you can LOAD or ENTER a new program, or to prevent someone from LISTing the contents of the current program in working storage. You can make NEW a program statement that will execute when the program finishes, preventing anyone from LISTing it after it has RUN. You can also use NEW in conjunction with ON ERR and ON IKEY to prevent the unauthorized LISTing of a program when an error or interrupt occurs.

How To Use It

You can use NEW as a program statement or as a keyboard command. Always do a NEW before LOADing or ENTERing a new program, unless you want to merge programs using ENTER (see ENTER in this chapter for merging techniques). If you use NEW as a program statement, make it the last executable statement, unless you only use it with ON ERR or ON IKEY.

What It Does

NEW clears all currently loaded statements and variables from working storage, leaving the working storage empty for new programs. It also closes any open file channels, and unlocks any locked files.

Examples

```
*NEW )

0005 ON IKEY THEN NEW

9999 NEW
```

- *Statement*
- *Command*
- *Function*

ON ERR

Trap an error in your program.

Format

ON ERR THEN statement

Arguments

statement	any legal BASIC statement except -- FOR, NEXT, DATA, MSG, END, REM, and DEF. The only commands you can use are -- SWAP, CHAIN, ENTER, SAVE, and REPLACE.

Why Use It?

Use ON ERR to direct your program to your own error handling routine instead of letting Business BASIC print an error message and stop program execution. For example, if you want to prevent the contents of a program from being LISTed if an error occurs, you could start the program with an ON ERR THEN NEW, and if an error occurred, working storage would be cleared.

What It Does

Normally when an error occurs during program execution, the program halts, the system prints the error message (from the BASIC.ER file -- see Appendix C), and your terminal returns to keyboard mode. If the system encounters an ON ERR statement before the error, the ON ERR statement tells the system what to do if an error occurs. If an error occurs after the ON ERR statement, the statement after THEN executes, and SYS(7) contains the error code for the appropriate error message. You can use SYS(7) and the ERM$ function to print the message that would have appeared if you hadn't used ON ERR.

Licensed Material-Property of Data General Corporation

ON ERR (continued)

How To Use It

You can only use ON ERR as a program statement. Since ON ERR will only trap errors that occur after it executes, you might want to place an ON ERR statement at the beginning of your program so that it will trap any error in the program. If you place the ON ERR statement anywhere else in your program, it will trap any errors that occur after it.

Examples

0010 ON ERR THEN GOSUB 0200

If error occurs, go to subroutine at 200.

0010 ON ERR THEN STOP

If error occurs, stop the program but don't print the error message.

```
0010 ON ERR THEN GOTO 0500
.
.
.
0500 DIM A$(50)
0510 LET A$=ERM$(SYS(7))
0520 PRINT A$
```

If error occurs, go to routine at 500 that dimensions A$, assigns the error message to A$, and prints it.

ON GOTO
ON GOSUB

Conditional transfer to other statements.

Formats

ON expr {GOTO / THEN} linenum$_1$ *[,linenum$_2$ ...]*

ON expr GOSUB linenum$_1$ *[,linenum$_2$ ...]*

Arguments

expr — a numeric expression that evaluates to an integer. If expr does not evaluate to an integer, it is truncated automatically as if you used the INT (Integer) function.

linenum — a valid line number of a statement or subroutine in your program that you want to execute. Line numbers range from 0 to 9999. You can make a list of line numbers, but only use 132 characters per line unless you change that with an STMA 4,8.

Why Use It?

Use ON GOTO or ON GOSUB to transfer control to one of several lines in a program. The line to which you transfer depends on the computed value of expr at the time the statement is executed. You can place these transfer points anywhere in a program, and use them to redirect program execution. Use ON GOTO (or ON THEN) to transfer control to another line, and ON GOSUB to transfer control to a subroutine which returns you to the line after the ON GOSUB.

Licensed Material-Property of Data General Corporation

ON GOTO
ON GOSUB (continued)

What It Does

ON GOTO and ON THEN do the same thing: expr evaluates to an integer, and control passes to -- the first line number in the ON statement when expr evaluates to 1, the second line number when expr evaluates to 2, the third line number when expr evaluates to 3, the tenth line number when expr evaluates to 10, and so forth. ON GOSUB works the same way except the line numbers refer to subroutines with RETURN statements that return control to the statement immediately following the ON GOSUB statement.

If expr is not an integer, expr is truncated to an integer (i.e., the fractional part is ignored). If the value of expr is greater than the number of line numbers you specify in the ON statement, or less than or equal to 0, then control passes over the ON statement and to the next statement. You can have up to 15 line numbers in an ON statement, unless you change the maximum page width with STMA 4,8.

Examples

```
0010 INPUT "TYPE MONTH (1-12):",M
0020 ON M GOSUB 0100,0200,0300,0400
0025 ON M-4 GOSUB 0500,0600,0700,0800
0030 ON M-8 GOSUB 0900,1000,1100,1200
0040 REM--ROUTINES RETURN TO HERE
.
.
.
0100 PRINT "JANUARY"
0110 RETURN
.
.
.
0200 PRINT "FEBRUARY"
0210 RETURN
.
.
.
0300 PRINT "MARCH"
0310 RETURN
.
.
.
```

In this example, if M=1 control passes to the subroutine at 0100; if M=2 control passes to 0200. If M>4, it falls through to the next ON GOSUB statement; if M>8, it falls through to the last ON GOSUB statement.

```
0005 RANDOMIZE
0010 LET X=RND(11)
0020 ON X GOTO 0100,0200,0300,0400,0500
0030 ON X-5 THEN 0600,0700,0800,0900,1000
0040 PRINT "X = 0"
0050 STOP
0100 PRINT "X = 1"
0120 STOP
0200 PRINT "X = 2"
0210 STOP
0300 PRINT "X = 3"
0310 STOP
0400 PRINT "X = 4"
0410 STOP
.
.
.
```

In this example, line 0020 and 0030 are equivalent statements. You can either use the form ON-GOTO or ON-THEN.

Licensed Material-Property of Data General Corporation

ON IKEY

Trap an interrupt in your program.

Format

ON IKEY THEN statement

Arguments

statement — any legal BASIC statement except -- FOR, NEXT, DATA, MSG, END, REM, and DEF. The only commands you can use are -- SWAP, CHAIN, ENTER, SAVE, and REPLACE.

Why Use It?

Use ON IKEY to direct your program to your own interrupt-handling routine otherwise Business BASIC can interrupt the execution of your program. Use ON IKEY THEN NEW as a security measure in RUN-only programs so in case an interrupt occurs, the program will not remain in working storage where other users can LIST it. It is also useful to put an ON IKEY statement before file I/O statements so that your file I/O will not be interrupted. The interrupt key is usually ESCape, Line Feed, or another key you or your system manager set, using STMA 4,6 or the TERM utility program.

What It Does

Normally when an interrupt occurs during program execution, the program halts, SYS(26) is set to 1, and your terminal returns to keyboard mode. If you've set a no-IKEY flag with STMA 6,5 and an interrupt occurs, the program doesn't know it -- it doesn't stop, the interrupt isn't handled by an ON IKEY -- only SYS(26) is set to 1 to indicate that an interrupt occurred. Under normal conditions, an ON IKEY handles any interrupt that occurs after the system encounters it. When an interrupt occurs after an ON IKEY statement, the statement after THEN is executed. If statement is a GOSUB, control passes to the subroutine and RETURN returns control to the statement immediately following the ON IKEY THEN GOSUB statement.

How To Use It

Use ON IKEY only as a program statement. Since ON IKEY only traps interrupts that occur after it has been executed, you might want to place an ON IKEY statement at the beginning of your program to trap any interrupt in the program. Place an ON IKEY elsewhere in your program and it traps interrupts that occur after it. You can restore normal handling of interrupts with an ON IKEY THEN INT statement, or with an ON IKEY THEN STOP statement. ON IKEY THEN INT and ON IKEY THEN STOP cancel the action of an ON IKEY THEN statement. If you disable interrupts with an STMA 6,5, then an interrupt will not be trapped in an ON IKEY, nor will it stop the program. You can then test SYS(26), the IKEY indicator, for a value of 1; if SYS(26) is 1, you can re-enable interrupts with an STMA 7,5, and the interrupt will then be trapped in the ON IKEY statement.

Examples

```
0005 ON IKEY THEN NEW
```

If an interrupt occurs, this statement will clear working storage.

```
0010 ON IKEY THEN PRINT "IKEY OCCURRED"
0020 STMA 6,5
.
.
```

(file I/O statements might go here)

```
.
.
0400 IF SYS(26)=1 THEN STMA 7,5
.
.
.
2000 STOP

*RUN )

IKEY OCCURRED

STOP AT 2000
*
```

The operator presses the interrupt key, but nothing happens while file I/O is going on, because line 20 set the no-IKEY flag. When system gets to line 0400, it tests SYS(26) to see if an interrupt occurred. Line 400 resets the no-IKEY flag to allow an interrupt to occur and print the message from line 10.

Licensed Material-Property of Data General Corporation

- *Statement*
- *Command*
  Function

OPEN FILE

Open a file or device in an access mode and give it a channel number.

Format

OPEN FILE (channel,mode),{"filename" / strvar}

Arguments

channel — a numeric expression for the channel you use to refer to the file in subsequent file I/O. You assign this channel number, in the range 0 to 15 (*not* 16 which refers to the terminal).

mode — a numeric expression for the access mode you assign to the file, in the range 0 to 7. We list the eight access modes further in "What It Does".

filename — a string literal in quotes (can be up to ten characters) for a disk file or device; a devicename (or link alias) must already exist in your directory.

strvar — a string variable that represents filename.

Why Use It?

You have to open a file and assign a channel and mode of access before trying to access it in file I/O statements. You can use OPEN FILE or the OPEN utility program (in Chapter 7) to do this. Read Chapter 4 before attempting to open files.

What It Does

OPEN FILE assigns a channel and a mode of access to the file or device specified by filename. You can then use channel in a file I/O statement to refer to this file or device. Table 3-2 is a summary of access modes.

Table 3-2. Access Modes

Mode	Type of Open	Type of I/O	Type of Access	Rule
0	Exclusive	Input/Output	Random	Create if file doesn't exist.
1	Exclusive	Output only	Sequential	Delete file, then create file.
2	Exclusive	Output only	Sequential	Append to existing file, or create file.
3	Shared	Input only	Sequential	File must exist.
4	Shared	Input only	Random	File must exist.
5	Shared	Input/output	Random	File must exist.
6	Exclusive	Input/output	Random	File must exist.
7	Exclusive use of mag or cassette tape	Input/output	MTDIO access	Tape must exist, tape file can be created, deleted or append to.

Licensed Material-Property of Data General Corporation

OPEN FILE (continued)

Mode 0

You can only open disk files in random mode. This opens the file exclusively for you (you don't need to LOCK it) and allows for both input and output. If the disk file named filename is not found in your directory, it creates filename in your directory as a random file.

Mode 1

You can either open a disk file or an output device for exclusive output. Mode 1 permits only WRITE FILE and PRINT FILE statements. If filename already exists in your directory, it deletes filename and creates a new filename (for devices, the last output is deleted and the device is ready for new output). If filename doesn't exist, it creates a new filename in your directory as a sequential file with 0 length.

Mode 2

You can open any output device or file in exclusive append mode. OPEN FILE positions the file pointer to the end of the current file so that you can append to it. If filename does not exist in your directory, a new filename is created as a sequential file. Mode 2 only allows sequential output.

Mode 3

You can open any input device or file in shared input mode. Filename must already exist in your directory or in $LIB directory. Only allows READ FILE or INPUT FILE access. Shared mode means that another user can open filename at the same time, so you might want to use file locking. Only sequential access is allowed.

Mode 4

You can only open disk files for shared random input. Random access allows you to position to a byte or a record within a random file. Filename must already exist in your directory or in $LIB directory. Only random input is allowed.

Mode 5

You can only open disk files for shared random access. In Mode 5 you can input from and output to a random file. Filename must already exist in your directory or in $LIB directory.

Mode 6

You can only open disk files for excusive random access. It allows you to input from and output to a random file, plus you have exclusive use of the file. Filename must already exist in your directory or in $LIB directory.

Mode 7

You can only open tape and cassette devices for direct tape access. It is exclusively opened for use with MTDIO (see MTDIO in this chapter).

How To Use It

You can only use modes 1, 2, and 3 with sequential files and devices, except mode 7 for tape devices. You can use any mode except mode 7 with random and modes 7 and 2 with contiguous files. Index files can only be opened in modes 0, 4, 5, and 6 (random access). Modes 0 and 2 create a file if it doesn't already exist. Mode 1 deletes the old file and creates a new one. Mode 2 appends to the file if it already exists. Modes 3, 4, 5, and 6 require filename to already exist. Mode 7 is used only for tape and cassette drives. Modes 0, 1, 2, 6, and 7 open the file exclusively for you so you don't need file locking, whereas in modes 3, 4, and 5 others can also open the file, thus you might need file locking. Read Chapter 4 thoroughly before deciding what kind of files you want and how you want to access them.

Examples

```
0010 OPEN FILE (0,1),"$LPT"
```

This statement opens line printer for output.

```
0100 OPEN FILE (1,3),"$PTR"
```

This opens paper tape reader for input.

```
*OPEN FILE (4,0),"DATA" )
```

```
0010 DIM A$(15)
0020 LET A$="FILE1FILE2FILE3"
0030 LET P=1
0040 FOR C=0 TO 2
0050   OPEN FILE (C,5),A$(P,P+4)
0060   LET P=P+5
0070 NEXT C
```


Licensed Material-Property of Data General Corporation

OR

Logical inclusive OR of two expressions.

Format

OR ($expr_1$, $expr_2$)

Arguments

$expr_1$ and $expr_2$ numeric expressions or variables you want compared.

Why Use It

OR is frequently used to check to see how many bits are set to 1 in a number.

What It Does

To illustrate how OR checks bits, we line up the bits for both exprs in Figure 3-4. For each bit, the result is a 1 if either expr has a 1 in that bit position; otherwise the result is a 0. If $expr_2$ is 0, the result will be $expr_1$ (the result of any value ORed to 0 or itself is always equal to the value).

How To Use It

If, for example, a status word (16 bits) was returned to you from MTDIO, first AND it with 32768 (2 ↑ 15) for the error bit (bit 15); if it's set (like the one in Figure 3-4), it means at least one other bit is set. You can find out how many are set by using OR with the status word and the value 32768. Subtract the result from 32768 (since you know that bit is set) and your remainder is the sum of the other bits set. In Figure 3-4, our result is 32802, and after subtracting 32768 from 32802 we have 34. Since the closest bit flag to 34 is 32 (2 ↑ 5 for bit 5), we now know that bit 5 (32) and bit 1 (2) are set. You can use the powers of 2 chart included with AND to find the appropriate values for bit flags. You can use OR as a numeric expression wherever numeric expressions are allowed.

BIT POSITION

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	$expr_1$
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	$expr_2$ Test value
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	Result

$expr_1$ = 32802 (status from MTDIO).

$expr_2$ Test value = 32768, or 2↑15 (status for "error" is bit 15 in status word from MTDIO).

Result = 32802, or bits 15, 5, and 1.

SD-01178

Figure 3-4. Logical OR of 2 Numbers

Licensed Material-Property of Data General Corporation

Examples

```
.
.
.
0030 MTDIO 1,1,REC$,ER%
0040 IF OR(ER%,32768)=ER% GOTO 0100
.
.
0100 LET X=OR(ER%,32768)-32768
0110 IF X 2 ↑ 8 THEN 0500
0120 LET X=OR(ER%,2)-2
0130 IF X=32 THEN PRINT "BAD TAPE"
.
.
.
```

Line 30 reads the tape record into REC$ and the tape status into ER%. If the error bit is set, control passes to line 100 where we subtract bit 15. If bits higher than 8 are set, control passes to line 500. The odd character bit is 2.

```
.
.
.
0050 READ FILE (2),REC$
0060 LET STAT%=ASC(REC$(1,2))
0070 LET STAT%=OR(STAT%,-AND(STAT%,32768))
.
.
.
```

The first 2 bytes of REC$ is the signed number STAT%. Line 70 corrects the sign of STAT%.

Licensed Material-Property of Data General Corporation

Statement
• *Command*
Function

PAGE

Set the page width of a job.

Format

PAGE = width

Arguments

width an integer expression in the range from 1 to 132 that specifies the number of columns (characters) allowed on a print line.

Why Use It?

Use PAGE to set your own page width on your terminal. By default the maximum number of characters (columns) per line is 72. You can set the page width from 1 to 132 characters.

What It Does

PAGE limits the number of characters permitted on a line.

How To Use It

PAGE can only be a keyboard mode command. Set width equal to the maximum number of characters you will allow on a line at your terminal. To set the page width in a program statement, use STMA 4,8.

Example

```
*PAGE=132 )
```

Statement
• *Command*
Function

POSITION FILE

Position the file pointer to a byte in a disk file.

Format

POSITION FILE (channel,byte)

Arguments

channel channel number of a file opened in mode 0, 4, 5, or 6 (for random access); a numeric expression.

byte numeric expression for the relative byte within a file to which you want to position the file pointer; can be something like: record-number * record-size.

Why Use It?

Use POSITION FILE to move the file pointer to a specific byte within a disk file when you want to read or write from that point. This is called "random access," and you'll find more on the subject in the "File Access" section of Chapter 4.

What It Does

POSITION FILE moves the file pointer to a specific byte relative to byte 0 of the file. Your byte expression may contain a record size multiplied by the number of records it takes to get the specific relative byte at which the record starts (POSFL.SL uses this form of POSITION FILE). POSITION FILE also resets the end-of-file flag for the EOF function. File input/output statements move the file pointer whenever they transfer data.

Licensed Material-Property of Data General Corporation

POSITION FILE (continued)

How To Use It

Use POSITION FILE if you want to move the file pointer fast and accurately. But you must know what relative byte you want, or you have to know the size of the fixed-length records and the number of the record you want. In some environments, exact byte locations are not known, and if you're using subfiles, it's easier to use POSFL.SL with the C1 array. You have to open your file for random access to use POSITION FILE.

Examples

```
0005 DIM REC$(100)
0010 OPEN FILE (0,5),"RANDOM"
0020 INPUT "RECORD NUM TO READ: ",RNUM
0030 LET RSIZE=100
0040 LET BYT=RNUM*RSIZE
0050 POSITION FILE (0,BYT)
0060 READ FILE (0),REC$
```

```
*OPEN FILE (1,0),"QUICK" )
*POSITION FILE (1,1084) )
```

(This command moves file pointer to byte 1084 in file QUICK.)

```
*DIM REC$(50) )
*READ FILE (1),REC$ )
```

This command reads 50 bytes (bytes 1084-1134) into REC$.

PRINT

Output characters to terminal, output file, or device.

Format

$$\left\{\begin{matrix}\text{PRINT}\\;\end{matrix}\right\} \textit{[FILE (chan),] [@ (num);] [TAB(n);] [expr]}$$

$$\left[\left\{\begin{matrix},\\;\end{matrix}\right\} \textit{expr} \ldots\right]$$

Arguments

chan numeric expression for the channel number of a file if you are outputting to a file or device; file must be opened for output.

@ (num) terminal control and cursor positioning expressions which can be repeated; they are described in "How To Use It."

TAB(n) TAB expression to tabulate to a given position *n* on a line; they can be repeated. They are described in "How To Use It."

expr numeric or string expression (variables, numbers, array elements, string literals, and string variables), numeric functions, but *not* string functions. You can repeat it. If you do not put *expr* in a PRINT (not PRINT FILE) statement, it will output a blank line.

Why Use It?

Use PRINT to output values to your terminal, and use PRINT FILE to output values to a file or device (like the line printer). The PRINT command is an excellent calculator in keyboard mode. Use PRINT to output string literals, strings contained in string variables, numeric expressions, and blank likes to your terminal. Use PRINT FILE to output characters (ASCII) that represent values of *expr* to the file, referenced by *chan,* that you opened for input.

Licensed Material-Property of Data General Corporation

What It Does

PRINT without FILE outputs the value of *expr* to the terminal. Each character takes up one column (space or position on a line). Numbers are printed with a leading space or minus sign and a trailing space. If you put a comma after expr, the next output starts at the beginning of the next tab zone (default is 14 columns, or set zones using the TAB statement). Put a semicolon after *expr* and the next output starts immediately following the output of *expr*. When the column counter is pushed by commas, semicolons, or the TAB expression past the 80 columns (or number of columns you specified in a PAGE statement) allowed on a terminal line, then an automatic carriage return/line feed moves the next output to the next line.

PRINT FILE outputs the character value of *expr* to the file specified by *chan*. If *chan* is 16, the file referenced is the terminal, thus PRINT FILE (16) behaves exactly as PRINT. Each *expr* is output as for PRINT, but PRINT FILE ignores terminal control (@) expressions (except the @ (-30) form feed). "Lines" of output to a file must terminate with a carriage return, form feed, or null: they can be up to 132 characters, at which point there is a carriage return inserted as a terminator. You can retrieve the characters output by PRINT FILE using INPUT FILE or INPUT FILE USING.

How To Use It

When you type a PRINT [FILE] statement or command, you can type a semicolon (;) in place of "PRINT" (e.g., "; EXPR","; FILE (0) EXPR").

You can set the tab zones at your terminal using the TAB statement, or accept the default tab setting (14 columns). See Figure 3-5.

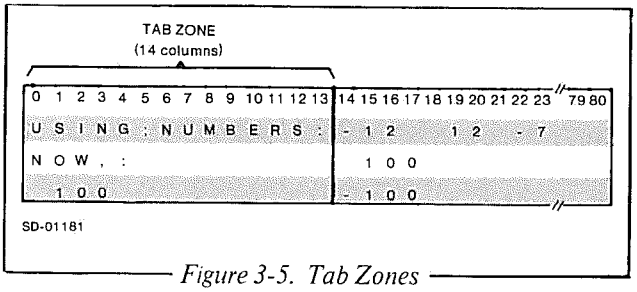


Figure 3-5. Tab Zones

To Print in Tab Zones

Use commas between each *expr* if you want to print in tab zones. A comma causes the next *expr* to be printed in the leftmost column of the next tab zone. PRINT checks to see if *expr's* output will fit on the line; if not, it will start printing in the first column of the next line (the next tab zone). A comma at the end of the list of *exprs* (i.e., at the very end of the PRINT statement) causes the next PRINT statement to output to the same line in the next tab zone; if there is no punctuation at the end of the PRINT statement, the next PRINT statement outputs to the next line. The PRINT command always puts output on the same line as the command. When printing triple precision numbers, remember that your number may be up to 15 digits but that is too big for the default tab zone (14 columns).

For a Compact Output

Use semicolons between each *expr* if you want compact output. A semicolon causes the next *expr* to be printed in the very next column. Characters only take up one column position, so if you want a space between characters, you must put that space in the string literal. Numbers take up one leading space or minus sign, the number of digits, and one trailing space, so you can put the next number immediately after the last, and there will still be a space between them. PRINT checks to see if *expr's* output will fit on the line; if not, it will start printing at the first column of the next line. A semicolon at the end of a PRINT statement (after all the *exprs*) causes the next PRINT statement to output to the same line in the very next column position; whereas, if there is no punctuation at the end of the PRINT statement, the next PRINT statement outputs to the next line. The PRINT command always puts output on the same line as the command.

To print a blank line

Use PRINT without *exprs* or punctuation, or use terminal control and cursor positioning (@) expressions, or use the TAB expression. All of these methods print a blank line.

Licensed Material-Property of Data General Corporation

The TAB expression

You can move the next *expr's* output to column *n* on the same line by using a *TAB(n)* expression, where *n* is an integer. Lines begin at column 0. Always make *n* a column number greater than the column where your last output ended; you can't move back on a line. If *n* is an integer greater than the page width (80 columns or set by the PAGE statement), *n* is reduced by the remainder after dividing *n* by the page width (*n*-MOD *(n,* width)). If *n* is 0 (i.e., TAB(0)), a carriage return/line feed occurs and the next output is printed in the first column (column 0) of the next line.

Terminal Control and Cursor Positioning

When using @ expressions, always separate them from the *expr* by following them with a semicolon, because a comma might put the cursor in the wrong position. We suggest that you also disable the column counter (STMA 6,3). Otherwise, the extra characters output by @ expressions may cause the line to appear as if it exceeds the current page width and cause an unwanted carriage return/line feed. PRINT FILE statements ignore all @ expressions except @(-30) which generates a form feed to skip to the next page. You can also use terminal control and cursor positioning expressions in INPUT [USING] statements.

Here are formats for terminal control and cursor positioning expressions:

@(line,column)	for cursor positioning.
@(item,value)	for terminal characteristics.
@(item)	for terminal control characters.

Cursor positioning

Use the format @(line,column), where line is the line number to which you want to position the cursor on the CRT (from 1 to 24), and column is the column position in line where you want to start the next output (from 1 to the page width for the CRT screen, or set by PAGE). This example

```
@(3,15)
```

positions to line 3, column 15.

Terminal characteristics

You can manipulate all terminal characteristics by STMA's, and each one is described in detail further in this chapter in STMA 4,-. We list each item below with the appropriate STMA reference. For terminal characteristics, use the format:

```
@(item,value)
```

where value is the ASCII value of the character you want to assign to item (see ASCII chart in Appendix A for the values), and item is one of the following (always negative):

-1 Line cancel key, see STMA 4,1.
-2 Character delete echo, see STMA 4,2.
-3 Character delete key, see STMA 4,3.
-4 Primary unpend key, see STMA 4,4.
-5 Secondary unpend key, see STMA 4,5.
-6 Primary interrupt key, see STMA 4,6.
-7 Secondary interrupt key, see STMA 4,7.
-8 Page width, see PAGE and STMA 4,8.
-9 Tab size, see TAB and STMA 4,9.
-10 Maximum number of characters allowed on input, see STMA 4,10.
-12 First-echoed character when line cancel occurs, see STMA 4,12.
-13 Second-echoed character when line cancel occurs, see STMA 4,13.
-14 Pad character, sent after all line feeds, see STMA 4,14.
-15 Number of pad characters to send, see STMA 4,15.
-19 Reset items -1 to -15 to their default values for your terminal. If you changed your terminal type with STMA 2,0, and you want the new terminal type's default characteristics, you must first do this reset.

Licensed Material-Property of Data General Corporation

PRINT (continued)

Terminal control characters

You can output before and after *exprs* in a PRINT statement (or before and after *prompt* in an INPUT [USING] statement) control characters to perform special CRT functions. Use the format:

@(item)

where item is one of the following (always negative):

-20 Move cursor to "home" (top line, first column of CRT screen).
-21 Move cursor right one column.
-22 Move cursor down one line.
-23 Move cursor left one column.
-24 Move cursor up one line.
-25 Sound the bell or audible alarm on the terminal.
-26 Tab to next field on screen.
-27 Return cursor to first column of current line.
-28 Move cursor to beginning of next line (new line).
-30 Clear entire screen (or generate a form feed in a file).
-32 Clear to end of the line.
-38 Start low intensity field (6053 terminals only).
-39 Start high intensity field (6053 terminals only).
-40 Start blinking field.
-41 End blinking field.
-52 Start underscored (underlined) field.
-53 End underscored field.

Examples

Here's an example of how to display a blinking underscored message:

```
*PRINT @(-40);@(-52);"HI! I'M YOUR CRT!";@(-53);@(-41) )
```

This example clears the screen and displays powers of 2 and their values in columns.

```
0010 PRINT @ (-30); @ (1,10);"POWER OF 2"; @ (1,30);"VALUE"
0020 FOR I=O TO 7
0030  PRINT @ (I+2,15);I; @ (I+2,30);2 ↑ I
0040 NEXT I
*RUN )

 POWER OF 2   VALUE
 0            1
 1            2
 2            4
 3            8
 4            16
 5            32
 6            64
 7            128

*PRINT "SUB";"STRING" ) SUBSTRING

0010 PRINT "TO ";"HAVE";"SPACES";
0020 PRINT "AND PRINT ON ONE LINE"
*RUN )
TO HAVE SPACESAND PRINT ON ONE LINE
*
```


Licensed Material-Property of Data General Corporation

```
0010 TAB=6
0020 FOR I=1 TO 11
0030   PRINT I,
0040 NEXT I
*RUN )
 1   2   3   4   5   6   7   8   9   10   11
*RUN )
*10 ) (Deletes TAB statement)
1       2       3       4       5       6       7       8       9       10       11
*

0010 DIM X$(255),AR(3)
0020 FOR I=0 TO 3
0030   INPUT "NUMBERS: ",AR(I);
0040 NEXT I
0050 INPUT "STRING: ",X$
0060 LET X=LEN(X$)+1
0070 PRINT X$;TAB(X);AR(0);TAB(2*X);AR(1);TAB(0);AR(2);TAB(X);AR(3);
0080 PRINT "DONE"
*RUN )
NUMBERS:  10 )  NUMBERS:  20 )  NUMBERS:  30 )  NUMBERS:  40 )
STRING:  21 CHARACTER SPACING: )
21 CHARACTER SPACING:10
30                              40                                      20
*

0010 DIM A$(132)
0020 OPEN FILE (0,0),"ASCIIDATA"
0030 INPUT FILE (0),USING "",A$,A$(LEN(A$)+1)
0040 IF EOF(0) GOTO 0300
0050 OPEN FILE (1,0), "NEWFILE"
0060 PRINT A$
0070 INPUT "IS DATA OK? YES(0) NO(1)",A
0080 IF A THEN GOTO 0500
0090 PRINT FILE (1),A$
0100 GOTO 0030
0300 CLOSE
.
.
```


Licensed Material-Property of Data General Corporation

- *Statement*
- *Command*
  Function

PRINT USING

Output characters to a terminal, file, or device according to a format.

Format

$$\left\{\begin{matrix}\text{PRINT}\\ \\ ;\end{matrix}\right\} \textit{[FILE (chan),]}\ \text{USING} \left\{\begin{matrix}\text{"formats"}\\ \\ \text{strvar}\end{matrix}\right\} \text{,expr } \textit{[,expr ...]}$$

Arguments

chan numeric expression for the channel number of a file or device opened for output; if outputting to file. If you make it 16, PRINT FILE (16) refers to terminal.

formats a string literal, in quotes, that contains any number of formats used to output expr; formats are described in "What It Does."

strvar a regular or subscripted string variable assigned the value of formats.

expr numeric or string expressions (variables, numbers, array elements, string literals, and string variables), numeric functions but *not* string functions; you can repeat expr with any combination of these.

Why Use It?

Use PRINT USING to output values using a special format or formats to your terminal, and use PRINT FILE USING to output values using a special format or formats to an output file or device like the line printer.

What It Does

PRINT [FILE] USING outputs values of exprs according to the instructions in formats. PRINT USING uses a buffer to formulate output. TAB expressions and semicolons are not permitted. PRINT FILE (16) USING... is the same as a PRINT USING statement. The formats you can use are:

A *n* Outputs *n* characters from a string expression in exprs. If the string is less than *n* characters, spaces are put to the right of the string. Nulls in the string are replaced by spaces.

D *w.d* Outputs a numeric expression in exprs using a total of *w* columns with *d* digits to the right of the decimal point. No decimal point is printed if *d* is 0. If the numeric expression is negative, the minus sign is placed in the leftmost position. Leading characters are suppressed.

E *w.d* Same as D *w.d* except the minus sign or space will be printed in the right-most position.

F *w.d* Same as D *w.d* except the minus sign is printed immediately to the left of the number and characters to the left of the sign are not filled.

K *w* Outputs a numeric expression in octal format using *w* columns.

L7 *c* or L7_*c* Sets the overwrite character to *c*. Any *c* in the buffer is replaced by numeric characters. The overwrite character is a space by default. The form L7_ *c* is required if *c* is a digit (in order to separate *c* from "7").

L8 *c* or L8_*c* Sets the fill character to *c* (fill character is a space by default). The fill character replaces leading zeros in D *w.d*, E *w.d*, and F *w.d* formats.

L9 Used in place of repeat count. The next expression in exprs is used as the repeat count to repeat use of formats.

L10 The current contents of the buffer is copied to a string variable in expr; the string variable must be DIMensioned to at least 132 bytes.

L11 Copies the string expression in expr into the buffer.

O Used immediately in front of D *w.d*, E *w.d* and F *w.d* formats to suppress printing if expr is zero.

P End the line in the buffer with a form feed instead of a carriage return (skip to top of next page on a 6053 CRT).

R *n* Same as A *n* except any fill characters required are placed on the left, making the output string right-justified.

S *n* Same as A *n* except no filling is done if the string is less than *n*.

Licensed Material-Property of Data General Corporation

PRINT USING (continued)

T *n* Set the column pointer to the *nth* position of the buffer. Note that due to previous PRINTs with the Z format or semicolon terminator, the internal column counter may be different from the column pointer used on the output device. If *n* is 0, the column pointer is set to the beginning of the last output field.

n X Move the column pointer *n* positions relative to where you are in the buffer; *n* may be negative.

Z End the line in the buffer with a null instead of a carriage return.

string Place the string directly into the buffer.

How To Use It

You can mix formats in one format string enclosed in quotes, or in a string variable for formats. You can repeat expr as many times as you like, but you have to repeat formats using a repeat count (a numeric expression outside the parentheses that enclose the formats) to repeat the instructions for each expr.

Examples

```
*PRINT USING "(' - -',TO,D11.0)",123456789 ) 123-45-6789

*PRINT USING "('JJJ JJ JJJJ',L8_0,L7J,TO,D11.0)",1234567 ) 001 23 4567

*PRINT USING "2(' ,',TO,D10.2)",123456, 123 ) 1,234.56 1.23

0010 DIM A$(40),FORMAT$(10)
0020 LET A$=" -",FILL$(32)
0030 LET FORMAT$="L11T0D8.0"
0040 PRINT USING FORMAT$,A$,5551212
*RUN )
555-1212

0010 OPEN FILE (0,0),"OUTPUT"
0020 PRINT FILE (0), USING "(30D1.0)",4,0,5
*RUN )
```

(A string of data would be output to OUTPUT: "405".)

```
*PRINT USING "3(OD1.0)",4,0,5 ) 4 5

*PRINT USING "'00:00:00',T0,L7_0,3(F2.0,1X)",SYS(14),SYS(13),SYS(12) ) 01:03:08
```


Licensed Material-Property of Data General Corporation

- *Statement*
- *Command*
Function

RANDOMIZE

Start/reseed the random number generator.

Format

RANDOMIZE

Why Use It?

Use it in conjunction with the RND function (see RND, later in this chapter), if you want the system to produce a random number. Normally, the RND function generates the same sequence of random numbers each time a program is RUN. Use the RANDOMIZE statement before the RND function when you want the random number generator to calculate different random numbers each time you RUN the program.

What It Does

RANDOMIZE reseeds the random number generator with a different base number according to the time or day, thereby producing different random numbers each time a program using the RND function is RUN.

How To Use It

You should start your program with a RANDOMIZE statement to initialize the random number generator; and you should execute the RANDOMIZE statement every time you want a different sequence of random numbers.

Example

```
0010 RANDOMIZE
0020 LET X=RND(11)
0030 PRINT X
0040 IF X=0 THEN STOP
0050 GOTO 0010
```

Line 20 picks a number in the range of 0 to 10. If the random number is 0, the program will stop.

- *Statement*
Command
Function

READ

Get values in DATA for variables.

Format

READ var_1 *[, var_2 ...]*

Arguments

var a numeric or string variable, depending on the value it will be assigned. The sequence of variables in READ must match the sequence of values in the data list formed by DATA statements. You must DIMension string variables and arrays greater than 11 elements (in any dimension) before use.

Why Use It?

Use READ in conjunction with DATA to initially assign values to variables quickly and easily (see DATA, this chapter). You can assign, for example, ten values to ten variables in a single READ/DATA combination, and you can change the values merely by changing the DATA statement.

What It Does

DATA statements store values in a data list before the program is executed. These values are assigned to the variables in READ one by one, sequentially. Values are retrieved from the data list in the order in which variables appear in the READ statement. A data element pointer is moved to the next available value in the data list as values are retrieved for variables in READ statements. If the number of variables in READ exceeds the number of values in the data list, an error occurs. Since READ variables and DATA values form lists no matter how many READ and DATA statements you use, the first value in the data list will always be the first value of the lowest numbered DATA statement, and the first variable in the variable list will always be the first variable in lowest numbered READ statement. If you want to restart the data list, use RESTORE (see RESTORE in this chapter).

Licensed Material-Property of Data General Corporation

READ (continued)

How To Use It

You can only use READ as a program statement. Normally, you place READ statements in a program at points where data is to be manipulated, whereas DATA statements can be placed anywhere in the program. You can subscript the numeric and string variables in READ, but values you use must match the corresponding values in the DATA statement, or an error will occur. READ is executable, so if you want to start READing variables at a specific point in your program, you can execute that specific READ statement.

Example

```
0010 DIM A$(10),ARA(12,12)
0020 READ X,Y
0030 READ A$,ARA(1,3)
0040 PRINT "X = ";X
0050 PRINT "Y = ";Y
0060 PRINT "A$ is : ";A$
0070 PRINT "ARA(1,3) = ";ARA(1,3)
0080 STOP
0090 DATA 32,46,"T. BOVE",1400

*RUN )

X = 32
Y = 46
A$ is : T. BOVE
ARA(1,3) = 1400

STOP AT 0080
*
```

READ FILE

Input binary data from a file.

Format

READ FILE (channel), var *[,var...]*

Arguments

channel — numeric expression for the channel number of a file opened for random or sequential access and input.

var — numeric or string variable, can be subscripted; can be repeated in any combination. Size of var determines the number of bytes input.

Why Use It?

READ FILE inputs binary data from any type of file. You determine the type of access when you open the file, but READ FILE will work for sequential and direct (random) access.

What It Does

The size of the var you supply determines the number of bytes input. For each var you supply in a READ FILE statement, a specific input occurs, so if you have five vars in a READ FILE, five separate inputs occur in sequential order. For each input, READ FILE reads in the number of bytes needed to fill the variable and moves the file pointer to the point where READing stops, so that the next input starts where the last one stopped. Single precision (%) numeric variables read in 2 bytes each, double precision (regular) numeric variables read in 4 bytes each, and triple precision (#) variables (if you're using them) read in 6 bytes each. Subscripted numeric variables (arrays) read in 4 bytes per array element for the range element for the range of elements in the subscripts. String variables read in their maximum length (i.e., the length specified in a DIM statement), and substrings read in the number of bytes specified in their subscripts (1 byte per character). If you read in bytes that were never written, you read in a null for each byte.

Licensed Material-Property of Data General Corporation

READ FILE (continued)

How To Use It

You can only read a file opened for sequential access from the beginning. You can read a file opened for random access from any byte position in the file. If your file is a subfile and/or dynamically allocated, then you have "records" that are fixed in length. A record is really the result of a read or write; but it is usually a fixed size in random files, and it must be a fixed size in dynamically allocated files, subfiles, and index files. When records are fixed in length, it is easy to learn what byte to position to -- just multiply the total number of bytes per record by the number of the record you want.

You can read part of a record into one or more vars, and you can read an entire record into one or more vars. Since each var in a READ FILE statement is a specific input, you might want to avoid an interrupt that occurs between inputs of a single READ FILE by using one large var in READ FILE (a single input cannot be interrupted). Use the EOF function to detect the end of the file. See "File Access' in Chapter 4 for more information.

Examples

```
0010 OPEN FILE (1,3),"$PTR"
0020 READ FILE (1),A,B,C,D,E,F,G
0030 PRINT A,B,C,D,E,F,G
0040 IF EOF(1) THEN 0020
.
.
.
```

This program opens the paper tape reader, and reads in 4 bytes for each variable. If an end of file occurs control passes to line 200.

```
0010 DIM REC$(48)
0020 OPEN FILE (2,0),"DATA"
0030 INPUT "RECORD?",R
0040 POSITION FILE (2,50*R)
0050 READ FILE (2),STAT%,REC$
0060 IF STAT% < = 0 THEN GOTO 0200
.
.
.
```

Line 50 reads the first 2 bytes into STAT%, and the rest into REC$. If the status of the record is less than or equal to 0, control passes to line 200.

Example

```
0010 DIM REC$(50)                 :Dimension string to 50 bytes (record size).
0020 OPEN FILE (2,0),"DATA"       :Open file DATA for random access.
0030 INPUT "RECORD YOU WANT TO READ: ",R
0040 POSITION FILE (2,50*R)       :Position to record R.
0050 READ FILE (2), REC$          :Read entire record into string.
0060 LET STAT%=ASC(REC$(1,2))     :Get first 2 bytes, a signed numeric, and
0070 LET STAT%=OR(STAT%,-AND(STAT%,32768))      :correct its sign.
0080 PRINT STAT%, REC$(3,50)      :Look at status and record.
.
.
.
```


Licensed Material-Property of Data General Corporation

- *Statement*
  Command
  Function

REM

A remark statement.

Format

REM text of remark

Arguments

text of remark the actual remark you want to make.

Why Use It?

Use REM statements to put certain comments in your program. They do take up space, however, so if you want in include extensive comments in your program, we suggest you use the comment mode available with the EDIT utility. Use EDIT to create your source file in ASCII format to hold the program and any extensive comments, while keeping a save file in binary format that holds the actual code without comments. This technique will save space when the program is running.

You should use REM statements at the entry points of subroutines for this reason: when you GOTO or GOSUB to a line that is a REM statement, the text of the REM statement is also put as a comment next to the GOTO or GOSUB that points to it. This labeling of GOTOs and GOSUBs will happen when you LIST the program, unless you have suppressed this feature using STMA 6,6.

What It Does

The text following REM is stored with the program and reproduced exactly when you LIST the program. Although REM statements are nonexecutable, note that storage space is required for the text.

How To Use It

REM can only be a program statement. Use it sparingly, and use it as the destination of a GOSUB or GOTO if you want your GOTO or GOSUB labeled, as in the example.

Example

```
*LIST )
0010 GOSUB 1000  : /THIS IS A LABEL
0020 GOTO 2000   : /THIS IS ANOTHER LABEL
.
.
.
1000 REM /THIS IS A LABEL
.
.
1050 RETURN
2000 REM /THIS IS ANOTHER LABEL
.
.
.
```


Licensed Material-Property of Data General Corporation

- *Statement*
- *Command*
  Function

RENAME

Change the directory entry for a file.

Format

RENAME { "oldname" / strvar } , { "newname" / strvar }

Arguments

oldname — string literal in quotes for a filename entry in your directory for a file that is in your directory or is a link entry.

newname — string literal in quotes for the new filename you want to give the file.

strvar — string variable that represents a filename.

Why Use It?

Use RENAME to rename any file but a SAVEd program file. SAVEd files carry their filenames within the files and you must use REPLACE to change their filenames not RENAME.

What It Does

RENAME looks for oldname, which is either a directory entry for a file in your directory or a link entry for a file in another directory, and changes it to newname. It doesn't disturb the contents of a file.

How To Use It

If you use strvar, you must DIMension it and supply a filename's value for it.

Examples

```
*RENAME "OLDDOG","NEWTRICKS" )

0010 DIM OLD$(10),NEW$(10)
0020 LET OLD$="KERMIT"
0030 LET NEW$="T.FROGG"
0040 RENAME OLD$, NEW$
```

- *Statement*
- *Command*
  Function

RENUMBER

Renumber lines in current program.

Formats

RENUMBER *[linenum]*

RENUMBER STEP increment

RENUMBER $linenum_1$ STEP increment

Arguments

linenum — First statement of program will be *linenum* and subsequent statements will be renumbered in increments of *linenum*. If you do not specify, *linenum* RENUMBER by itself will start at line 10 and increment by 10's.

increment — First statement will be line number 10 and subsequent statements will be incremented by the increment value.

$linenum_1$ — First statement will be $linenum_1$ and subsequent statements will be renumbered by increments of increment.

Why Use It?

Use RENUMBER to renumber the statements of your program, usually so that you have more room to insert new statements.

What It Does

RENUMBER will renumber every statement in your program, and will renumber the destination line numbers of GOTO, GOSUB, and ERASE statements to coincode with the renumbered lines. Let's look at each form of RENUMBER:

RENUMBER

Starting at line 10, renumber all lines in increments of 10.

Licensed Material-Property of Data General Corporation

RENUMBER (continued)

RENUMBER linenum

Starting at linenum, renumber all lines in increments of linenum.

RENUMBER STEP increment

Starting at 10, renumber all lines in increments of increment.

RENUMBER $linenum_1$ STEP increment

Starting at $linenum_1$, renumber all lines in increments of increment.

How To Use It

Make your line numbers integers greater than 0 and less than or equal to 9999. If you execute a RENUMBER that would exceed that limit, the system instead performs an automatic RENUMBER 1 STEP 1.

Examples

```
*RENUMBER )
0010 REM -- THIS PROGRAM COMPUTES BALANCE
0020 INPUT "TYPE BALANCE: $",BAL
0030 IN BAL = 0 THEN STOP
0040 INPUT "AMOUNT OF CHECK: $",CHK
0050 IF CHK = 0 THEN STOP
0060 LET BAL = BAL-CHK
0070 PRINT "BALANCE IS: $",BAL
0080 GOTO 0040

*RENUMBER 20 )
0020 REM -- THIS PROGRAM COMPUTES BALANCE
0040 INPUT "TYPE BALANCE: $",BAL
0060 IF BAL = 0 THEN STOP
0080 INPUT "AMOUNT OF CHECK: $",CHK
0100 IF CHK = 0 THEN STOP
0120 LET BAL = BAL-CHK
0140 PRINT "BALANCE IS: $",BAL
0160 GOTO 0080

*RENUMBER STEP 5 )
0010 REM -- THIS PROGRAM COMPUTES BALANCE
0015 INPUT "TYPE BALANCE: $",BAL
0020 IN BAL = 0 THEN STOP
0025 INPUT "AMOUNT OF CHECK: $",CHK
0030 IF CHK = 0 THEN STOP
0035 LET BAL = BAL-CHK
0040 PRINT "BALANCE IS: $",BAL
0045 GOTO 0025

*RENUMBER 5 STEP 5 )
0005 REM -- THIS PROGRAM COMPUTES BALANCE
0010 INPUT "TYPE BALANCE: $",BAL
0015 IF BAL = 0 THEN STOP
0020 INPUT "AMOUNT OF CHECK: $",CHK
0025 IF CHK = 0 THEN STOP
0030 LET BAL = BAL-CHK
0035 PRINT "BALANCE IS: $",BAL
0040 GOTO 0020
```


Licensed Material-Property of Data General Corporation

- *Statement*
- *Command*

  Function

REPLACE

Replace a program on disk with a new program.

Format

REPLACE [{ "*filename*" }]
        [{ *strvar* }]

Arguments

filename a literal filename to which you want to change the save file's name; optional.

strvar string variable already DIMensioned and assigned a *filename* value.

Why Use It?

REPLACE is the only way you can replace an old program with a new one under the same save filename. You can rename the new program to a new *filename* if you specify *filename,* and it will not delete the old one. REPLACE will, however, delete *filename* if it already exists, and create the new program as a save file named *filename.*

What It Does

REPLACE performs a SAVE on whatever program you have in working storage. REPLACE saves the program under whatever *filename* you specify, or the current program name if you don't specify *filename.* If a program already exists with that *filename,* the program is replaced with the new one. If you didn't LOAD or RUN the old program, and you don't specify a new *filename* for the new one, the new program is saved with the filename, "SCRATCH".

How To Use It

Use REPLACE as a keyboard mode command or a program statement. Either way, it will save the current program. If you LOAD or RUN a program so that it is in working storage, then you can update the program and REPLACE the old program with the updated one, and either change its name by specifying a new *filename* with REPLACE (thereby keeping the old program intact with its name), or keep the same name by not specifying *filename* with REPLACE (thereby destroying the old program). If you do *not* LOAD or RUN a program, but create a new one in working storage, you can REPLACE any program with this new program by specifying the old program's name as *filename* with REPLACE. If you do not specify *filename* for this new program, the new program will be saved by REPLACE under the filename "SCRATCH".

Note: A program's filename is carried internally within the save file. If you RENAME a save file (RENAME is a BASIC CLI command, find it in Chapter 8), the old name still exists within the save file. Therefore, if you LOAD and then REPLACE a save file without giving a *filename,* the program will be saved under its old filename.

Examples

```
*REPLACE )
```

if you LOADed or had RUN the program, then modified it, this would save the modified one under the old name. If you just created a new program in working storage, this would save it under the name SCRATCH.

```
*REPLACE "PROG3" )
```

Current program is saved under new name PROG3. If PROG3 already exists, it is deleted and this new one takes its place.

```
0900 LET NAME$="NEWPROG"
0910 REPLACE NAME$
```

Current program is saved under new name *NEWPROG* replacing any program of the same name.

Licensed Material-Property of Data General Corporation

- *Statement*
- *Command*
  Function

RESTORE

Reset DATA data list pointer.

Format

RESTORE *[linenum]*

Arguments

linenum optional argument for a valid line number in your program, can be in the range 0 to 9999. If you include *linenum*, RESTORE resets the data list pointer to the first value for the DATA statement at that *linenum*.

Why Use It?

When you want a READ statement to go back to the beginning of a DATA statement data list to pick up values for its variables, use RESTORE (see READ and DATA in this chapter).

What It Does

RESTORE without *linenum* resets the pointer to the beginning of the data list. The first element in the data list is the first value in the lowest numbered DATA statement. RESTORE with *linenum* resets the pointer to the first value in a specific DATA statement, specified by *linenum*. If you specify a *linenum* that doesn't exist in your program or is not a DATA statement, RESTORE finds the next DATA statement following *linenum* and resets the pointer to the first value in it.

How To Use It

Use RESTORE as a program statement or a keyboard mode command. If you want to reuse the entire data list, don't specify *linenum*. If you want to reuse the values in a particular DATA statement, specify the DATA statement by its *linenum*.

Example

```
0005 READ A,B,C
0010 READ D,E,F
0015 RESTORE 0040
0020 READ G,H,I
0025 RESTORE
0030 READ J,K,L
0035 DATA 2,4,6
0040 DATA 8,10,12
```

In the above example, the variables are assigned values as follows:

Variable	Value
A	2
B	4
C	6
D	8
E	10
F	12
G	8
H	10
I	12
J	2
K	4
L	6

Licensed Material-Property of Data General Corporation

Statement
Command
• *Function*

RND

Produce a pseudo-random integer.

Format

RND(expr)

Arguments

expr a numeric expression or variable such that the random integer generated will be greater than or equal to 0 and less than (but not equal to) expr.

Why Use It?

The RND function produces a different set of random numbers each time you call it and use it with the RANDOMIZE statement. Without RANDOMIZE, you can generate a fixed series of random numbers.

What It Does

Each time you use the RND function, it produces a number *n* such that 0 = *n* = expr -1. If you do not use RANDOMIZE, and you run the program again, you will get the same sequence of random numbers. RANDOMIZE restarts the sequence at a different starting point, so you get a different sequence of random numbers each time you use RANDOMIZE with RND.

How To Use It

RND can be a numeric expression wherever numeric expressions are allowed. If you want to repeat a series or random numbers, don't use RANDOMIZE.

Examples

```
0010 FOR I=1 TO 4
0020   PRINT RND(100)
0030 NEXT I
*RUN )
21
26
54
90
```

Run same program again.

```
*RUN )
21
26
54
90
```

Now add RANDOMIZE statement

```
*5 RANDOMIZE )
*RUN )
43
86
14
23
*
```


Licensed Material-Property of Data General Corporation

Statement
• *Command*
Function

RUN

Execute a program.

Format

RUN [{ *"filename"* / *linenum* }]

Arguments

linenum a statement line number of the current program in working storage where execution begins.

filename a literal filename, in quotes, for a save file on disk or a binary input device, which will be LOADed and executed. You cannot use *linenum* with *filename.*

Why Use It?

Use RUN to execute a program either from the first line or from a specified line. RUN with *filename* performs a NEW (clear the working storage), then a LOAD to bring the program into working storage, then executes the program from the first line. CHAIN works the same way, but you can use CHAIN as a program statement. SWAP also LOADs and executes a program, but when the new program stops, the old program returns to working storage (see CHAIN, SWAP, and LOAD in this chapter).

What It Does

The variations of the RUN command are:

RUN

Clears all variables, undimensions all arrays and strings, does a RESTORE, initializes the random number generator, and then executes the current program from the first line number.

RUN *linenum*

Retains current values for variables, retains file status, maintains file position pointers, keeps locks active (do *not* do a RESTORE), and executes the current program beginning at *linenum.* If *linenum* is the line following the program's point of interruption, this form is equivalent to CON.

RUN *"filename"*

Searches directory for this save file; if not found, searches library directory for *filename.* Performs a NEW, LOADS *filename,* and then executes *filename* from the first line (*filename* is now in working storage).

How To Use It

You need to type RUN with *"filename"* if you want to RUN a program; otherwise, *filename* by itself performs a SWAP. Use the RUN *linenum* form to help debug your programs.

Examples

```
*RUN )
```

(This executes the current program from the first line.)

ERROR 31 AT 0050--SUBSCRIPT

```
*RUN 0050 )
```

(This retains current values for variables, and executes program starting at line 50.)

ERROR 31 AT 0050--SUBSCRIPT

```
*RUN "NEWPROG" )
```

(This clears working storage, LOADs NEWPROG, and executes NEWPROG from the first line.)

Licensed Material-Property of Data General Corporation

- *Statement*
- *Command*
  Function

SAVE

Save a program on disk.

Format

SAVE { "filename" / strvar }

Arguments

filename a literal filename for your new save file. It should be a new file; if filename already exists, you'll get an error message.

strvar a string variable already DIMensioned and assigned a filename value.

Why Use It?

Use SAVE to save a program on disk in core image binary format. A program in core image binary format takes up less space than an ASCII source copy of the program, and you can LOAD a SAVEd program faster than you can ENTER a LISTed ASCII-format program. SAVEd programs can be CHAINed to, SWAPped to and REPLACEd. You can also SAVE a program which has partially executed; at some later date you can resume execution (with a CON or RUN *linenum*) of the program while retaining the values the variables had when you SAVEd it.

What It Does

SAVE will write the current program (source statements and data in working storage) in binary format to a disk file it creates using the name filename. Filename must not already exist in your directory, or you'll get an error message. Filename is stored internally in the save file, so if you want to rename your save file, you should use REPLACE, *not* RENAME. SAVE will not preserve the file status of files in your program.

How To Use It

You supply the SAVE filename in quotes or a string variable that holds a value of filename. When you SAVE a program, the current values of all variables are stored with the program as well as the location where the program stopped last. Therefore, when an interrupt occurs, you can LOAD the program and CONtinue execution of it as if no interruption has occurred. To replace a saved program with a new (perhaps edited version of the old) program, use REPLACE (see REPLACE in this chapter) but *don't* use RENAME because RENAME does not change the internal name of the save program. LOAD your program, edit it, then use REPLACE (without a filename to keep the same name, with a filename if you want to change the name to filename). you can also RUN filename if filename is the name of a save file (i.e., RUN will do an implicit LOAD).

Examples

```
*SAVE "OURSHIP" )
```

This saves current program in a save file called "OURSHIP".

```
0900 LET NAME$="PROG3"
0910 SAVE NAME$

*SAVE "PROG3" )
I/O ERROR 9- FILE ALREADY EXISTS
*REPLACE )
```

The file "PROG3" was already saved, so we replace the old one with the current program, and keep the same name.

Licensed Material-Property of Data General Corporation

Statement
Command
• *Function*

SGN

Determine the sign (positive or negative) of an expression.

Format

SGN(expr)

Arguments

expr a numeric expression or variable for the number whose sign you want to know.

Why Use It?

A frequent use of SGN is to test for the sign of a number in a conditional transfer statement.

What It Does

SGN returns a value of 1 if expr is positive, a value of -1 if expr is negative, and a value of 0 if expr is 0.

How To Use It

Use SGN as a numeric expression wherever numeric expressions are allowed. SGN should only be used to check the sign of a number or numeric variable -- if you extracted a 2-byte numeric from a string, its sign would be incorrect (the sign of a 4-byte numeric transfer is correct). See the second example to see how to correct the sign of a 2-byte number extracted from a string.

Examples

```
0010 INPUT X
0020 IF SGN(X)=-1 THEN PRINT "NEGATIVE"
0030 IF SGN(X)=1 THEN PRINT "POSITIVE"
0040 IF SGN(X)=0 THEN PRINT "EQUALS ZERO"
.
.
.


.
.
.
0050 READ FILE (2),REC$
0060 LET STAT%=ASC(REC$(1,2))
0070 LET STAT%=OR(STAT%,-AND(STAT%,32768))
0075 IF SGN(STAT%)=-1 THEN GOTO 0800
.
.
.
```

Line 60 extracts STAT% from string REC$. Line 70 corrects the sign of STAT%. If STAT% is negative, control passes to line 800.

SHFT

Logical 32-bit shift of a value's bits.

Format

SHFT(expr,bits)

Arguments

expr numeric expression or variable; number you want to be shifted.

bits number of bits you want to shift. A negative (-bits) shifts bits number of bits to the right (less), and a positive (+bits) shifts bits number of bits to the left (greater).

Why Use It?

When you use AND or OR, you sometimes use values that are the equivalent to setting one bit (e.g., 32768 is bit 15). If you want to check for bits 15-12, one at a time, you could just shift the value 32768 of bit 15 -1 bits (to the right), and shift it -1 bits again (to the right) and it becomes the value of bit 13 (2 ↑ 13), etc.

What It Does

SHFT with a negative number in bits shifts expr bits number of bits to the right. All the bits of expr move bits number of bits to the right. SHFT with a positive number in bits shifts expr bits number of bits to the left. All the bits in expr move bits number of bits to the left. SHFT with a 0 in bits does nothing. See Figure 3-6.

How To Use It

Use SHFT as a numeric expression wherever numeric expressions are allowed. Our example uses SHFT with AND to check the status word returned by MTDIO. If bit 15 is set, we test for bits 14, 12, 10, 9, 8, 7, 5, or 1 by shifting the test value V.

BIT POSITION

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X = 32768, or 2↑15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	SHFT(X, -1) = 2↑14
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	SHFT(X, +2) = 2↑16

SD-01182

Figure 3-6. The SHFT Function

Example

```
.       :Code to open mag tape file, set up parameters, etc., goes here.
.
.
0540 MTDIO 1,8,ER%              :Read tape status into ER%.
0545 LET V=2↑15                 :Test first for error (bit 15).
0550 IF AND(ER%,V) THEN GOTO 1000       :If error, go to 1000.
.
.
.
1000 LET V=SHFT(V,-1)           :Test for bit 14.
1010 IF AND(ER%,V) THEN GOTO 2000       :If bit 14 is set, go to 2000.
1020 LET V=SHFT(V,-2)           :Test for bit 12.
1030 IF AND(ER%,V) THEN GOTO 3000       :If bit 12 is set, go to 3000.
1040 LET V=SHFT(V,-2)           :Test for bit 10.
1050 IF AND(ER%,V) THEN GOTO 4000       :If bit 10 is set, go to 4000.
1060 IF AND(ER%,SHFT(V,-1)) THEN GOTO 6000      :If bit 9 is set, go to 6000.
1070 IF AND(ER%,SHFT(V,-2)) THEN GOTO 7000      :If bit 8 is set, go to 7000.
.
.
.
```


Licensed Material-Property of Data General Corporation

Statement
• Command
Function

SIZE

Determine the size of program in working storage.

Format

SIZE

Why Use It?

Use SIZE to find out how many bytes of working storage the current program takes up, and how many bytes are still available.

What It Does

SIZE outputs to the terminal the number of bytes by the program and the number of bytes still available in working storage. The values for the number of bytes are decimal.

How To Use It

SIZE can only be a keyboard mode command. Just type SIZE and see what happens.

Example

```
*SIZE )
USED: 6700 BYTES
LEFT:  8077 BYTES
```

Statement
Command
• Function

SQR

Square root (truncated to be integer) of an expression.

Format

SQR(expr)

Arguments

expr numeric expression or variable you want to find the square root of.

Why Use It?

Use it to find the square root of any number. If you want precision in your answer (since we only have integer arithmetic), multiply your expr by the power of 100 for the precision you want.

What It Does

SQR finds the square root of expr and truncates it making the value an integer.

How To Use It

Use SQR as a numeric expression wherever numeric expressions are allowed. If you want more precision, for example, to 3 decimal places, multiply expr by 100 ↑ 3. See the example.

Examples

```
0010 LET X=2
0020 PRINT SQR(X)
0030 PRINT
0040 LET X=X*100 ↑ 3
0050 PRINT USING "D5.3",SQR(X)
*RUN )
1

1.414
*
```


Licensed Material-Property of Data General Corporation

- *Statement*
- *Command*
  Function

STMA

User system call.

Format

STMA type,arguments

Arguments

type — a numeric expression or number designating the type of system call. Each type has its own format and arguments. We discuss each type separately below.

arguments — Each type has its own format and arguments, usually numeric or string expressions or variables, separated by commas.

Why Use It?

STMA is a very powerful statement. It is a system call that allows you to examine and modify many aspects of your job. You can examine and change terminal characteristics, terminal types, and examine account information; all of these are set in the HELLO program that uses STMAs. You can pass information between programs, check for IKEYs (interrupts) and status flags, set echo/no echo and message/no message flags, reset FOR/NEXT and GOSUB/RETURN stacks if you've nested them up to their limits, and here's a great feature: you can translate any code into your code (or ASCII default) on input and output, as well as do ASCII/EBCDIC translations.

Here are the STMAs: each one has type as its first number, and each has its own format and available items.

STMA 1, item, variable

Where:

variable is a simple numeric variable that receives a value when you execute the call.

item is a number indicating one of the available items:

0 Terminal type (set by HELLO or by yourself at log-on).

1 Error code passed between programs that are using SWAPs and/or CHAINs.

2 Available for you to pass or retrieve information between programs (one-word allocations).

3 Same as 2. You might need more than one place to pass or retrieve information.

When you execute an STMA 1, your variable must be already initialized (i.e., given a value, even if you give it a zero). After the STMA execution, your variable will hold the value you're looking for. The variables in STMA 1 and STMA 2 pass 16-bit values, or one-word allocations. Here are some examples:

STMA 1,0,X — gives you the terminal type in X.

STMA 1,1,ER — gives you the error code passed to your program from another program that used STMA 2,1 and a SWAP or CHAIN to your program. Error code will be in ER.

STMA 1,2,VAR or STMA 1,3,VAR — gives you the data in VAR that is passed to your program from another program that used STMA 2,2 or STMA 2,3 and a SWAP or CHAIN to your program.

STMA 2, item, value

Where:

item is a number indicating one of the available items for STMA 1.

value is a numeric expression for the value you want the item to be set to. Here are some examples:

```
STMA 2,0,6
or
LET T=6
STMA 2,0,T
```

This will set your terminal type to terminal type 6. Note: If you set a new terminal type, you must do a PRINT @ (-19) to set the default terminal characteristics for the new type.

Look at the example in Figure 3-7 of the program called "HOME" that passes an error code (using SYS(7)) and data using STMAs to the program "REMOTE" which receives them.

Licensed Material-Property of Data General Corporation

In "HOME", if an error occurred on input, the error code would be in SYS(7) (see the SYS function in this chapter), and an error message would appear at the terminal. To pass this error to the other program, you just have to pass the value of SYS(7). The input value is passed using VAL and retrieved in D. These are 2-byte transfers, so if you want to pass ASCII values, you can only pass one ASCII character's code at a time.

Example

```
0010 REM --THIS IS HOME
0020 ON ERR THEN STMA 2,1,SYS(7)          :If error, pass the error code.
0030 INPUT USING "",VAL                   :Incorrect input will cause error.
0040 STMA 2,2,VAL                         :Pass data in VAL.
0050 SWAP "REMOTE"


0010 REM --THIS IS REMOTE
0020 DIM ERR$(30)                         :Initialize error string.
0030 LET D=0                              :Initialize variables.
0040 LET ER=0
0050 STMA 1,1,ER                          :Get error code, if any.
0060 IF ER<>0 THEN GOSUB 0100             :If error, go to routine at 100.
0070 STMA 1,2,D                           :Get data passed from HOME.
0080 PRINT D                              :Print data.
0090 STOP
0100 REM --ROUTINE TO PRINT ERROR.
0110 LET ERR$=ERM$(ER)                    :Use ERM$ function to get message.
0110 PRINT ERR$                           :Print message.
0120 RETURN
```

Figure 3-7. HOME SWAPs to REMOTE

STMA 3, item, variable

This retrieves the value of item and puts it in variable.

STMA 4, item, value

This sets the value of item from value.

Where:

variable is a simple numeric variable that receives the value of item.

value is a numeric expression for the value you want to set the item to (usually an ASCII value).

item is a number indicating one of the available items:

0 De-attach key. Striking this key while executing a job will cause the job to de-attach from the terminal and leave the terminal free for more program development. The job will continue to run in a de-attached state. See ATTACH, a utility program described in Chapter 7.

1 Line-cancel key. Strike this key to cancel an erroneous line you just typed.

2 Character-delete echo. This character is sent to the terminal whenever you delete a character. If you set it to a value greater than 127, then the sequence (value -128),space,(value -128) will be echoed on the terminal. This is how it appears to erase a character: the cursor backs up, prints a space, and backs up again, so you'd set your value to the ASCII decimal value of a backspace plus 128.

3 character-delete key. Usually set to the RUBOUT or DELETE key.

4 Primary unpend key. Usually a carriage return, indicated in our examples by the symbol ")". See explanation for the secondary unpend key (next) for an explanation of both.

Licensed Material-Property of Data General Corporation

STMA (continued)

5 Secondary unpend key. Whenever a program stops on an INPUT statement, it is waiting on an answer. An "unpend" key terminates your input. For example, when you key a statement or command line, you usually end the line with a carriage return (), because the carriage return (ASCII <13>) is the default primary unpend character. Whenever a program stops running, the primary unpend character (no matter what you set it to be) is set to carriage return and the secondary unpend character is set to a nonenterable character (ASCII <255>). It is automatic so that you never have the problem of forgetting what key to hit when you want to unpend. When the program stops, you can set the unpend keys to whatever you want, and unpending will occur if you strike one of those keys. The unpend key is converted to a carriage return before it is placed in the I/O buffer.

In an INPUT statement is terminated by a secondary unpend key, SYS(10) will equal 1 (see the SYS function in this chapter). If you set both unpend keys to the ASCII value <255> (nonenterable character), then unpending will occur on any character you input and the character will be put in the I/O buffer (useful for allowing only one-character inputs). When you set both unpend keys to <255>, control characters and lowercase characters are not ignored, and line-cancel and character-delete keys are not processed. Unpend characters are never echoed on the terminal.

NOTE: Function keys on a 6053 terminal are actually a two-character sequence, where the header character is <30> for all 11 keys and the second character is a lowercase character from q-z for each key. You can use one as a secondary unpend key by setting the secondary unpend key to <30>.

6 Primary interrupt key. This is usually the ESCape key, but can be any key you choose.

7 Secondary interrupt key. This can also be any key you choose. Interrupts will interrupt a program's execution (see Chapter 2 for more details about interrupts) and return you to keyboard (command) mode, unless you handle the interrupts in your program with ON IKEY statements (see this chapter for ON IKEY). There are also ways to test whether or not an interrupt occurred without actually enabling interrupts -- see STMA 5,6, and 7, and SYS(26). SYS(10) will tell you what kind of interrupt occurred: A 0 value is a primary interrupt, a 1 value is a secondary interrupt.

8 Page width, or number of characters allowed on a line at your terminal. The page width can't be longer than 132. See PAGE in this chapter for more information.

9 Tab size. This sets equal-size tabs on a line for spacing your output. A tab cannot be longer than the current PAGE width setting. See TAB in this chapter for more information

10 Maximum number of characters allowed on input. This allows you to fix the amount of characters a terminal operator can type on an input. Any additional characters are not echoed and are not stored. It is always reset to 132 whenever the program stops. If you set it to a negative number, the absolute value of the number will be the number of characters allowed on input, and unpending will occur on the last character. Therefore, an STMA 4,10,-1 is the same as the sequence STMA 4,4,255 and STMA 4,5,255.

11 Not used.

Licensed Material-Property of Data General Corporation

Example

```
0005 DIM A$(512)
0010 LET VAL=0              :Initialize variable used in STMA's.
0020 STMA 3,0,VAL           :Find out value of the de-attach key.
0030 PRINT VAL
0040 INPUT "NEW VALUE FOR DE-ATTACH KEY: ",N
0050 STMA 4,0,N             :Set new value for de-attach key.
0060 STMA 3,10,VAL          :Find out max number of char allowed on input.
0070 STMA 4,10,-20          :Allow 20 char on input, and unpend on the 20th.
0080 INPUT A$               :Will only allow 20 char to be input, and will
.                           :unpend on the 20th or on carriage return if you
.                           :input less than 20.
.
```

12 First-echoed character when a line cancel occurs. This is usually set to a backslash (\), which is an ASCII <92>. When you hit the line-cancel key, you usually get two echoed characters: the first-echoed character is the backslash (\), and the second-echoed character is usually a carriage return (ASCII <13>) to return the cursor to the beginning of a new line.

13 Second-echoed character when a line cancel occurs. This is usually a carriage return (ASCII <13>). See explanation for first-echoed character above (STMA 4,12).

14 Pad character. This character is sent after all line feeds so that you can't type anything before the cursor returns to the beginning of the line on slow terminals.

15 Number of pad characters to send. Normally you don't need any pad characters for fast terminals, like DG 6052's and 6053's.

STMA 5, flag, variable

This retrieves the value of flag (0 or 1) and puts it in variable.

STMA 6, flag

This sets flag; flag's value is 1, or "on".

STMA 7, flag

This clears flag; the flag value is 0, or "off".

Where:

variable is a simple numeric variable that receives the value of flag and must be initialized before use.

flag is one of the following available status flags:

0 No echo. Also called a half-duplex. Input characters are not send back (echoed) to the terminal.

1 Allow lowercase characters to be input. Normally, lowercase alphabetic characters are converted to uppercase characters.

2 Allow control characters to be input. Normally, control characters (except line and character delete) are ignored on input.

3 Disable the column counter. Normally when enough characters have been output to equal the page width, BASIC inserts a carriage return/line feed to put the cursor at the beginning of the next line. Setting this flag inhibits that action.

4 No messages allowed. Can only be overriden by an OPCLI command.

5 No IKEY. This flag will not allow an interrupt to stop your program, nor to be caught in an ON IKEY trap. It will only allow the IKEY indicator, SYS(26), to be set to 1 to indicate that an interrupt occurred. If you cleared this flag and SYS(26) equaled 1, then an interrupt would occur immediately. You reset SYS(26) by issuing an STMA 8,3.

6 Suppress listing of labels on GOTOs and GOSUBs. If the destination of a GOTO or GOSUB is a REM of the form -- REM \ *comment*, then the *comment* will appear after the GOTO or GOSUB when the program is LISTed. This always occurs when you GOSUB to one of the Business BASIC subroutines described in Chapter 7, because each subroutine begins with a REM statement of the form above. To keep this from happening, set this flag.

Licensed Material-Property of Data General Corporation

Example

```
0010 LET VAR=0                          :Initialize variable for STMA's.
0020 STMA 5,4,VAR                       :Find out if no-message flag is set.
0030 IF VAR=1 THEN STMA 7,4             :If set, clear it.
0040 STMA 5,5,VAR                       :Find out if no-IKEY flag is set.
0050 IF VAR=0 THEN STMA 6,5             :If not, then set it.
.                                       :Code for file I/O goes here.
.                                       :I/O will not be interrupted.
.
0100 IF SYS(26)=1 THEN STMA 7,5                 :If an interrupt occurred, then
.                                               :re-enable interrupts to process IKEY.
.
.
```

STMA 8,0

This resets the FOR/NEXT stack and the GOSUB/RETURN stack. If you nest FOR/NEXT loops or GOSUB/RETURN routines beyond their legal nesting limits, you will get ERROR 18-GOSUB NESTING or ERROR 20-FOR NESTING. You can then reset the stacks and start all over again, or write new nested loops and routines after this point.

STMA 8,1

This resets only the FOR/NEXT stack.

STMA 8,2

This resets only the GOSUB/RETURN stack.

STMA 8,3

This resets the SYS(26) IKEY indicator. Use it in conjunction with STMA 6,5 and the SYS function, described in this chapter. Normally you use STMA 8, 3 to test if an interrupt occurred without enabling interrupts.

STMA 8,4

This clears, or "flushes" the input buffer. This is useful in heavy key-entry environments where operators can key ahead rapidly. If an operator makes an error, you'd want to flush the input buffer before making the operator re-key the information; otherwise you'd have garbage in your input buffer.

STMA 9, item, strvar

Where:

strvar is a string variable (already DIMensioned) that receives the string value of item.

item is one of the following available items:

0 Account specifier. A 6-character string you set in the ACNTS file in the HELLO program.

1 Current directory name. Up to 10 characters set by the HELLO program or a CLI DIR command.

2 Current program name. Up to 14 characters set by a SAVE or REPLACE command.

3 Current default output device, usually $LPT for the line printer. Set by CLI command SQUE.

STMA 10, 0, string

This sets the special characters allowed in crammed strings -- CRM$/UCM$ conversions.

Where:

string is a string literal in quotes or a string variable already DIMensioned and assigned a string value.

When you use the CRM$ function, you "cram" three bytes of a string into two bytes of the resulting string. You can "cram" and "uncram" a string accurately if the string has any of these characters: A-Z (not lowercase), 0-9 and four special characters, usually a comma (,), a dash (-), a period or decimal point (.), and a space. These are the default special characters, but you can specify your own CRM$/UCM$ special characters with this STMA. Your string, as either a string literal in quotes or a string variable, must contain the special characters you want to specify. See the CRM$ and UCM$ functions in this chapter.

Licensed Material-Property of Data General Corporation

STMA (continued)

Examples

```
0010 DIM A$(4)
0020 SET A$="+-$="
0030 STMA 10,0,A$
```

Or:

```
0010 STMA 10,0,"+-$="
```

Both examples define the special characters as a plus (+), a minus (-), a dollar sign ($), and an equal sign (=). You can now use these characters instead of the default characters in CRM$/UCM$ conversions.

STMA 11, julian-date, var_1, var_2, var_3

This converts a julian date to month, day, and year.

Where:

var_1	receives the value for month.
var_2	receives the value for day.
var_3	receives the value for year.
julian-date	is a numeric expression or variable for the number of days since 1/0/68 (January 0, 1968).

All variables must be initialized before using them is an STMA statement or command.

STMA 11 converts a julian date into the current month, day, and year. A julian date is the number of days from a base year, and we use Data General's birth year as the base year: January 0, 1968. If you use this as your base year, the julian date will be correct through February 29, 2100.

Example

```
0010 LET MONTH=0
0020 LET DAY=0
0030 LET YEAR=0
0040 LET JULIAN=3761
0050 STMA 11,JULIAN,MONTH, DAY, YEAR
0060 PRINT MONTH;"/";DAY;"/";YEAR
```

This program will print the result: 4/18/78, the date translated from the julian date 3761.

STMA 12, variable, month, day, year

This converts a date of the form "month/day/year" to a julian date.

Where:

variable	receives the value for the julian date (days since January 0, 1968).
month, day, and year	are numeric expressions or variables for the date you specify.

You must initialize all variables before using them in STMAs.

STMA 12 converts a date of the form "month/day/year" to a julian date (number of days since Jan. 0, 1968). The julian date is returned in variable. You can then get the day of the week by using the MOD function with variable: MOD (variable,7) would return a 0 for Sunday, 1 for Monday, and so forth.

Licensed Material-Property of Data General Corporation

Example

```
0010 LET JULIAN=0                    :Initialize variable JULIAN.
0020 INPUT "MONTH: ",MONTH
0030 INPUT "DAY: ",DAY
0040 INPUT "YEAR: ",YEAR             :Input dates.
0050 STMA 12,JULIAN,MONTH,DAY,YEAR    :Convert to a julian date.
0060 LET D=MOD(JULIAN,7)              :Get day of week using MOD function.
0070 ON D GOTO 0100,0110,0120,0130,0140,0150
0080 PRINT "THIS IS A SUNDAY"         :If D=0.
0090 STOP
0100 PRINT "THIS IS A MONDAY"         :If D=1.
0105 STOP
0110 PRINT "THIS IS A TUESDAY"        :If D=2.
0115 STOP
0120 PRINT "THIS IS A WEDNESDAY"      :If D=3.
0125 STOP
0130 PRINT "THIS IS A THURSDAY"       :If D=4.
0135 STOP
0140 PRINT "THIS IS A FRIDAY"         :If D=5.
0145 STOP
0150 PRINT "THIS IS A SATURDAY"       :If D=6.
0155 STOP
```

STMA 13, strvar, tablestring

This translates a string of characters using a table you set up.

Where:

strvar — is a string variable that contains the string to be translated and receives the translation.

tablestring — is another string variable or string literal in quotes containing the table with a maximum of 256 characters.

You must DIMension string variables before using them.

In the computer, characters exist as code numbers. We use the ASCII code set (see Appendix A), where every character has a matching code number. We include control characters, upper/lowercase, special symbols, even notprintable characters; there are 256 in all. Since there are other codes out in the real world, and since you'll probably need data from another machine that uses a different code, we've implemented STMA 13 to translate any string of characters into any code you want. For standard ASCII/EBCDIC and upper/lowercase translations, see STMA 14.

When you read in the foreign data, each character has a unique binary code number, and you must know the character set and decimal equivalents. Let's say the first character in strvar has a code number of 5. STMA 13 would count 5 positions into tablestring, using 1 (not 0) as a base, and arrive at position 6 in tablestring. The code 5 character would translate to a value you previously put in tablestring at position 6. The result of the translation would then be put in the first position of strvar. STMA 13 allows a character with a code of 0 to translate to the first position in tablestring; that's why you count "code + 1" positions into your tablestring and at that place puts your desired character.

If the characters you input to STMA 13 are ASCII characters, STMA 13 uses ASCII code to count into tablestring. Your tablestring can accommodate 256 characters. If the code of your input character is greater than 256, the character translates to the character in the last position of tablestring. Any character in strvar whose code is greater than the length of tablestring translates to the last character in tablestring.

Examples

```
*10 DIM DATA$(50),TABLE$(256) )
*20 LET TABLE$="1234567890" )
*30 LET DATA$="<2><5><8>" )
*40 STMA 13,DATA$,TABLE$ )
*50 PRINT DATA$(1,1),DATA$(2,2),DATA$(3,3) )

*RUN )
 3  6  9

*
```


Licensed Material-Property of Data General Corporation

STMA (continued)

In the above example, the character with a code of 2 translates to a "3," a character with a code of 5 translates to a "6," and a character with a code of 8 translates to a "9." In the next example, the program allows any ASCII character input to translate to the same ASCII character except the "&" sign (ASCII <38>) translates to a comma (ASCII <44>):

```
0010 DIM DATA$(256),TABLE$(256)
0020 FOR J=0 TO 255
0030  LET TABLE$(J+1)=CHR$(J)
0040 NEXT J
0050 LET TABLE$(39,39)=CHR$(44)
0060 INPUT USING "", DATA$
0070 STMA 13,DATA$,TABLE$
0080 PRINT DATA$
```

For ASCII codes 0 to 255, line 30 fills the table with ASCII values. Line 50 changes "&" to a comma in the table. Line 60 allows input of data to be translated on line 70.

STMA 14, strvar, type

This translates a string of characters according to a pre-set table: ASCII/EBCDIC, EBCDIC/ASCII, and lowercase ASCII/uppercase ASCII.

Where:

strvar is a string variable (must be DIMensioned before use) that contains the string to be translated and receives the translation.

type is a number indicating one of the following available translation types:

- 0 Lowercase alphabetic ASCII characters to uppercase alphabetic ASCII characters only.
- 1 ASCII code to EBCDIC code.
- 2 EBCDIC to ASCII.

STMA 14 is a special case of STMA 13 that provides the standard conversions needed in a business environment. For translations other than the types above, use STMA 13.

Example

```
0010 OPEN FILE (0,7), "MT0:0"
0020 DIM DATA$(80)
0030 MTDIO 0,1,DATA$,E%
0040 STMA 14,DATA$,2
0050 OPEN FILE (1,2), "OUTPUT"
0060 PRINT FILE (1),DATA$
.
.
.
```

This program reads data from tape in EBCDIC format, and converts EBCDIC to ASCII.

STMA 15, $string_1$, $string_1$, variable

This compares two strings.

Where:

$string_1$ and $string_2$ are string variables or string literals in quotes, or string expressions.

variable is a simple numeric variable that receives either a "match" or "no-match" value:

- 0 not a match
- 1 a match

The characters in $string_1$ are compared with those in $string_2$, returning a value of 1 in variable if they match, or 0 if they don't match. The strings must be equal in length and correct character by character for a match to occur, unless you use dash (-) and asterisk (*) conventions, only allowed in $string_1$. Here is a summary of the dash and asterisk conventions:

- A dash (-) matches any number of characters including 0 characters, but does not match a decimal point (used also for filename extensions). Here are examples:

$string_1$	**$string_2$**	
"-AB-"	"CLASSAB"	Match!
"-AB-"	"FABRST"	Match!
"-AB-"	"CABR.ST"	No match -- forgot extension.
"-AB-.-"	"CABRST.UVWXYZ"	Match!

Licensed Material-Property of Data General Corporation

STMA (continued)

- An asterisk (*) matches any character which is in the same place as the asterisk. Only one character matches per asterisk. Here are examples:

$string_1$	$string_2$	
"*AB**"	"CABRS"	Match!
"*L*B***"	"AL BOVE"	Match!
"*CD**"	"RSTCDEF"	No match.
"*CD-"	"RCDEFGHIJ"	Match!
"-CD-.*"	"UVWCDRST.A"	Match!
"-CD-.*"	"UVWCDRS.TA"	No match.
"******"	"1234"	No match -- $string_1$ greater.

Example

```
0005 LET NUM=0
0010 DIM A$(20),B$(20)
0020 OPEN FILE (0,0), "MYFILE"
0030 READ FILE (0),A$
0040 OPEN FILE (1,0), "YOURFILE"
0050 READ FILE(1),B$
0060 STMA 15, A$, B$, NUM
0070 IF NUM=0 THEN PRINT "NO MATCH"
0080 IF NUM=1 THEN PRINT "MATCH!"
.
.
.
```

STMA 16, value

This de-attaches the current job.

Where:

value is a number, numeric expression, or variable that has a value of 0 or 1 as follows:

0 When the current job de-attaches, the terminal remains inactive (HELLO program is not called) until you log on again.

1 When the current job de-attaches, it calls the HELLO program that automatically starts the log-on procedure at your terminal.

Use STMA 16 in your program to de-attach your job; make value a 0 -- if you want to leave the terminal inactive until someone logs on by hitting the ESCape key, or a 1 -- if you want the HELLO program to start log-on automatically.

Use STMA 16 and STMA 17 in your program to de-attach the program and attach to another port. A large print program can de-attach itself from your terminal and attach itself to a receive-only port that has no keyboard and can't be logged on. See STMA 17 for an example.

STMA 17, port, error

This attaches current job to port.

Where:

port is a number, numeric expression, or variable indicating a terminal number (use STAT or PORTS to find one).

error is a numeric variable, already initialized, that will receive the error code in an error occurs.

STMA 17 attaches the current job to the terminal indicated by port leaving the current terminal inactive. The current terminal can be logged on again. If port already has a job attached to it, or if port is invalid, STMA 17 will return an error code in error. This is useful for starting a print job on a receive-only terminal which cannot be logged on otherwise.

Example

```
0010 LET ERROR=0
0020 STMA 16,1
0030 STMA 17,5,ERROR
0040 IF ERROR THEN SWAP "STAT"
.
.
.
```

Line 20 de-attaches the job, and calls HELLO to log on automatically. Line 30 attaches the job to port 5. If there is an error, call the STAT program.

Licensed Material-Property of Data General Corporation

STMA (continued)

STMA 18, filename, variable

This examines, assigns or frees a reserved file or device.

Where:

filename is the name of a reserved file or device (reserved files are set up during BASIC system generation).

variable is a numeric variable that returns a specific value when you assign a specific value to it; the available values are:

0 To examine a device or reserved file's condition. It returns a 0 in variable if filename is free; otherwise, it returns a number greater than 512 which is the base address in the user status table of the job that has the file reserved.

1 To free filename from the job executing the STMA so that other jobs can use the file or device. If the call is successful, it returns a 1 in variable; otherwise it returns a value greater than 512 in Variable -- the base address in the user status table of the job that has the file reserved.

2 To assign filename to the job executing the STMA so that no one else can access that device or file. If the call is successful, it returns a 2 in variable; otherwise it will return a value greater than 512 in variable -- the base address in the user status table of the job that has the file reserved.

NOTE: If filename is not a valid reserved file or device, STMA 18 returns a value of -1 in variable.

STMA 18 is used to control access to reserved files and devices. Device names and reserved files in your directory can be found by doing the BASIC CLI command LIST/A -- using the /A global switch to list permanent as well as nonpermanent files.

Example

```
0010 LET VAR=0                      :Initialize variable.
0020 STMA 18,$LPT,VAR               :Examine the line printer.
0030 IF VAR>512 THEN GOTO 0010      :If not free, try again.
0040 PRINT "DEVICE READY"
0050 LET VAR=2
0060 STMA 18,$LPT,VAR               :Assign line printer to me.
0070 IF VAR=2 THEN GOTO 0090        :If successful, continue the program.
.                                   :Error code goes here.
.
0090 REM -- PRINT ROUTINE
.
.
.
0300 REM -- END OF PRINT ROUTINE
0310 LET VAR=1
0320 STMA 18,$LPT,VAR               :Free the line printer.
0330 IF VAR=1 THEN GOTO 0400        :If successful, go to end of program.
0340 PRINT "RESERVED FILE NOT FREED, WILL TRY AGAIN."
0350 GOTO 0310
.
.
.
```


Licensed Material-Property of Data General Corporation

- *Statement*
- *Command*
- *Function*

STOP

Stop program execution.

Format

STOP

Why Use It?

Use STOP to halt the execution of a program at some point. Usually a program ends when it runs out of statements, or when it encounters an END statement. STOP is more convenient if you want the program to stop at a certain point, and it is useful for debugging programs.

What It Does

When STOP is encountered, the program stops, and a message appears at your terminal:

STOP AT *xxxx*

where *xxxx* is the line number of the STOP statement. The terminal is put in keyboard mode and it waits for your next command.

How To Use It

Use STOP as a program statement only. It can be its own statement, or it can be a statement after IF/THEN or ON ERR THEN or ON IKEY THEN. You can put STOP statements anywhere in a program.

Example

```
0010 ON ERR THEN STOP
0020 ON IKEY THEN STOP
0030 INPUT X
0040 IF X=0 THEN STOP
0050 PRINT X
0060 STOP
```

- *Statement*
- *Command*
- *Function*

SWAP

Execute a utility or another program and return.

Format

SWAP { "filename" / strvar } [*THEN* { *GOTO linenum* / *CON* }]

Arguments

filename — a literal filename in quotes, for a save file on disk, *not* an ASCII listing or source file. File must contain a BASIC program or Business BASIC utility program.

strvar — a string variable already DIMensioned and assigned a filename value.

linenum — a valid line number in the program to which you want to SWAP. Execution begins at *linenum* rather than the beginning of the program, and the program retains the values it had when it was SAVEd.

Licensed Material-Property of Data General Corporation

SWAP (continued)

Why Use It?

Use SWAP to execute another program or a Business BASIC utility program from the currently running program, and when the new program stops, you are returned to the current program. SWAP is used more often then CHAIN to execute utility programs because you usually want to be returned to the original program after the utility has completed. Utility programs are discussed in Chapter 7.

What It Does

SWAP works like CHAIN, but note that SWAP does *not* clear the current program from working storage and you return to the current program after the SWAP. Using SWAP THEN CON causes the new program to start at the point where it last stopped, retaining the variables' values as if a CON occurred (see CON in this chapter). Using SWAP THEN GOTO *linenum* causes the new program to start at *linenum* and retains the values the variables had when the program was SAVEd, as if a RUN *linenum* had occurred (see RUN in this chapter).

SWAP searches for filename in your directory; if not found, it searches the library directory; if still not found, it gives you an error message. SWAP does not change the status of files.

How To Use It

Use SWAP as a program statement or a keyboard mode command. As a keyboard mode command, the keyword SWAP is optional: just type the filename in quotes. If you use it as a program statement, SWAP executes the new program and when the new program stops, it will return you to the original program that had the SWAP statement. If you use SWAP as a keyboard mode command, it executes the new program and returns you to keyboard mode. This is the way the BASIC CLI works when you type filename as a command to the BASIC CLI.

Examples

```
0900 SWAP "PROG102" THEN GOTO 0100
0910 PRINT "PROG102 FINISHED"
```

Start executing PROG102 at line 100 and then return to line 910.

```
*SWAP "PROG3" THEN CON )
```

This executes PROG3 at the point where it last stopped.

```
*"INDEXBLD" )
```

This SWAPs to INDEXBLD utility program.

```
*"PROG3" THEN GOTO 200 )
```

This SWAPs to PROG3 and starts at line 200.

Statement
Command
● *Function*

SYS

Return system information.

Format

SYS(item)

Arguments

item numeric expression or variable for one of the items listed in "What It Does."

Why Use It?

Use SYS to get specific system information you use as a value in your program.

What It Does

SYS returns the information indicated by item, where item is one of the following:

0 the time of day since midnight (seconds past 00:00).
1 the day of the month (1-31).
2 the month of the year (1-12).
3 the year in 4 digits (e.g., 1978)
4 the terminal port number (-1 if de-attached).
5 CPU time used, in seconds to the nearest tenth of a second.
6 I/O usage (number of system calls made).
7 the error code of the last or current error, used in conjunction with the ERM$ function and the BASIC.ER file, see Appendix C.
8 the channel number of the file most recently referenced in a file I/O statement.
9 job number.
10 alternate IKEY or unpend flag; set to 1 if you use the alternate IKEY or unpend key.
11 global switches used when the Business BASIC system was brought up.
12 time of day (seconds past the last minute).
13 time of day (minutes past the last hour).
14 time of day (hours since midnight).
15 status word 1 (from user status table).
16 status word 2 (from user status table).
17 program length in bytes.
18 data length in bytes.
19 maximum program and data length.
20 statement number of last error or IKEY.
21 current CPU switch settings.
22 time remaining after last INPUT or TINPUT.
23 log on time (minutes past 00:00).
24 current push level.
25 processor number (0=single, 1=master, 2=slave, 3=secondary single).
26 status of IKEY flag (1 if IKEY occurred). Reset by STMA 8,3.
27 time of day as *hhmmss* (*hh* is hours, *mm* is minutes, *ss* is seconds).
28 data as *mmddyy* (*mm* is month, *dd* is day, *yy* is year).
29 unique number generated in a series, no two jobs ever get the same number.

NOTE: The value of SYS(7) in invalid if no runtime error occurred, and the value of SYS(8) is invalid if no file was opened.

How To Use It

Use SYS as a numeric expression wherever numeric expressions are allowed. If you don't understand the explanation given or the value received, look up each item. Some of the information is privileged for operators and system managers, so you might not find further explanations in your manual.

Licensed Material-Property of Data General Corporation

Examples

```
0010 FOR I=0 TO 29
0020   PRINT SYS(I)
0030 NEXT I

0005 ON ERR GOTO 1000
.
.
.
1000 PRINT "DO YOU WANT TO SEE ERROR MESSAGE?"
1010 INPUT "YES(0),NO(1):",A
1020 IF A THEN GOTO 1040
1025 LET X$=ERM$(SYS(7))
1030 PRINT X$
1040 GOTO 1200
.
.
.

*PRINT SYS(27) ) 234804

*PRINT SYS(28) ) 051478

*PRINT SYS(2)"/";SYS(1);"/";SYS(3) ) 05/14/1978

*PRINT SYS(14);":";SYS(13);":";SYS(12) ) 23:52:31
```


Licensed Material-Property of Data General Corporation

TAB

Set the tab width.

Format

TAB = width

Arguments

width an integer expression greater than 0 and less than or equal to the width of PAGE (default PAGE width is 72, see PAGE).

Why Use It?

Any output by a PRINT, PRINT FILE, or PRINT USING statement can make use of tabulated print zones, whether they output to the terminal, line printer, or an ASCII file. The default TAB width is 14 characters, which allows 5 tabulated print zones in a 72-character ine (72 is the default PAGE width). You set your own TAB width with this command.

What It Does

TAB sets tab stops on a line, allowing width number of characters between them. Output from PRINT separated by commas falls into the newly tabulated print zones. See PRINT in this chapter for more information and for terminal control of output.

How To Use It

TAB can only be a keyboard mode command. To set print zones in a program statement, use STMA 4,9. Set your width equal to the number of characters allowed in a print zone.

Example

```
*TAB=10 )
*PRINT 1,2,3 )
 1   2   3
*
```


Licensed Material-Property of Data General Corporation

- *Statement*
- *Command*
  Function

TINPUT

Perform an INPUT in a specified amount of time.

Format

TINPUT time { *[FILE (chan),]* / *["prompt"]* } *[USING "",]* var *[,var ...]*

Arguments

All the arguments to TINPUT can be found in INPUT [FILE] and INPUT [FILE] USING. Follow the same rules, except:

time — numeric expression for the number of tenths of a second to wait for input.

Why Use It?

TINPUT is most frequently used to allow only a certain amount of time for input from the terminal. Log-on programs usually use TINPUT to fix a certain amount of time for input -- if you don't answer the question in time, it logs you off.

What It Does

TINPUT performs an INPUT. TINPUT USING performs an INPUT USING. TINPUT FILE performs an INPUT FILE. The only difference is that if the input has not completed before time is up, the statement after TINPUT is executed (or if you're using the TINPUT command, you are put back in keyboard mode). If the variables in TINPUT are not filled when time is up, they remain unchanged.

How To Use It

Specify time as a numeric expression for seconds in tenths of a second (e.g., 100 is 10 seconds). The function SYS(22) gives you the remaining time on the current TINPUT, and if SYS(22) equals 0, the current TINPUT has timed out.

Examples

```
0010 DIM A$(132)
0020 TINPUT 100,USING "","INPUT DATA: ",A$      :10 seconds allowed only.
0030 IF SYS(22)=0 THEN GOTO 0100
0040 PRINT USING "('YOU TOOK',D4.2,' SECONDS TO INPUT: ',S80)",100-SYS(22)
0050 PRINT A$
.
.
.
0100 PRINT "TIMEOUT.  TRY AGAIN."
0110 GOTO 0020
.
.
.
```


Licensed Material-Property of Data General Corporation

Statement
Command
• *Function*

TRUN$

Truncate a string.

Format

strvar$_1$ = TRUN$(strvar *[,1]*)

Arguments

strvar$_1$ string variable or substring you want to receive truncated string.

strvar string variable or substring you want truncated, may be the same as strvar$_1$.

1 optional "1" causes TRUN$ to truncate at the last nonspace character in the string.

Why Use It?

TRUN$ without the *1* is frequently used to truncate large strings read in from a file or device when the large string has inbedded carriage returns, form feeds, or nulls. TRUN$ with the *1* is frequently used to shorten strings that have been lengthened with nulls or spaces.

What It Does

TRUN$ (strvar) truncates the string at the first null, form feed, or carriage return and assigns the truncated string to strvar$_1$. If strvar$_1$ and strvar are not the same, nothing happens to strvar after the truncation but note that strvar$_1$ receives the truncated version of strvar.

TRUN$ (strvar, *1)* truncates strvar when it reaches the last nonspace character, or when it reaches a null, form feed, or carriage return. It does this by going as far as the first form feed, null, or carriage return and moving backwards to the first nonspace character, at which point it truncates strvar.

How To Use It

Use TRUN$ in assignment statements (or the LET command) only, because you have to assign the truncated string to strvar$_1$. Use TRUN$ without the *1* to reduce your string to its current length if it is padded to the end with nulls. Use TRUN$ with the *1* to shorten a string that is padded with nulls or spaces to the end, or to cut off trailing spaces.

Examples

```
*DIM A$(512) )
*LET A$="ABC DEF GHI ",FILL$(0) )
*PRINT LEN(A$) ) 512

*LET A$=TRUN$(A$) )
*PRINT LEN(A$) ) 13

*LET A$=TRUN$(A$,1) )
*PRINT LEN(A$) ) 11

*
```


Licensed Material-Property of Data General Corporation

UCM$

Uncram a crammed string.

Format

$strvar_1$ =UCM$(strvar)

Arguments

$strvar_1$ string variable or substring to receive uncrammed string; can be the same as strvar.

strvar string variable or substring (already crammed) you want to be uncrammed.

Why Use It?

CRM$ shortens a string to save space, and UCM$ is the only way you can "unshorten" it correctly. If, for example, you received the value of a crammed key from an index file, you'd have to uncram the key string, using UCM$, in order to print the string.

What It Does

Every 2 bytes of strvar are uncrammed to 3 bytes and put in $strvar_2$, which must be sufficiently large enough to hold the uncrammed version of the string, or else you lose some of the string. You can only unpack the following characters: A-Z (*not* lowercase), 0-9, and four special characters, set either by using STMA 10 or the default characters -- comma (,), decimal point (.), minus sign (-), and space. Any character in strvar that is not in this set is uncrammed to a space.

How To Use It

You can only use the UCM$ function in assignment statements (or the LET command) because you must assign the uncrammed string to $strvar_1$. If $strvar_1$ and strvar are the same, then you lose the crammed version of the string when you perform the UCM$ on it. If strvar has not been cremmed using CRM$, you'd get a spaced-out string that is difficult to interpret.

Examples

```
0010 DIM X$(6),Y$(9)
0020 LET Y$="ABCDEFGHI"
0030 LET X$=CRM$(Y$)
0040 LET Y$(1,6)=X$
0050 LET Y$=UCM$(X$)
0060 PRINT Y$
*RUN )
ABCDEFGHI
*

*DIM A$(9),B$(6) )
*LET A$="AB@CD$EF." )
*LET B$=CRM$(A$) )
*PRINT LEN(B$) ) 6

*LET A$=UCM$(B$) )
*PRINT A$ ) AB CD EF.

*
```


Licensed Material-Property of Data General Corporation

- *Statement*
- *Command*
  Function

WRITE FILE

Output binary data to a file.

Format

WRITE FILE (channel), var *[, var ...]*

Arguments

channel numeric expression for the channel number of a file opened for random or sequential output.

var numeric or string variable, can be subscripted; can be repeated in any combination. The size you make var determines the number of bytes that are output.

Why Use It?

WRITE FILE outputs binary data to any type of file. You determine the type of access when you open the file, but WRITE FILE will work for sequential and random (direct) access.

What It Does

The size you specify for var determines the number of bytes output to the file. For each var in a WRITE FILE statement, a specific output occurs; so if you have five vars in a WRITE FILE, five separate outputs occur in a sequential order (i.e., each output is put after the preceding one in the file). The file pointer is moved to the end of the last output, so the next output starts where the last one stopped. Single precision numeric variables (%) write out 2 bytes each, double precision (regular) numeric variables write out 4 bytes each, and triple precision numeric variables (if you're using them) write out 6 bytes each. Subscripted numeric variables (arrays) write out 4 bytes per array element for the range of elements in the subscripts. String variables write out their DIMensioned length, whether or not they are filled with actual data (empty string bytes write out as null bytes). Substrings write out the number of bytes specified in their subscripts (1 byte per character).

How To Use It

You can only write to a file opened for sequential access from the beginning and sequentially from that point. You can write to a file opened for random access starting at any byte position in the file, writing sequentially from that point. If your file is a subfile and/or dynamically allocated, then you have "records" that are fixed in length. A record is a name we use for fixed-length areas in a dynamically allocated file, subfile, or index file. When records are fixed in length, it is easy to learn what byte to position to -- just multiply the total number of bytes per record by the number of the record you want to write.

You can write part of a record using one or more vars, and you can write an entire record using one of more vars. If you are rewriting a record, you should write an entire record so that you rewrite every byte in the record. Since each var in a WRITE FILE statement is a specific output, you might want to avoid an interrupt that occurs between outputs of a single WRITE FILE by using one large var in a WRITE FILE (a single output cannot be interrupted).

Before attempting to use files, you should read Chapter 4 thoroughly to find out what kinds of files best suit your needs. WRITE FILE is also described in the "File Access" section of Chapter 4.

Examples

```
0010 OPEN FILE (0,1), "FBI"
0020 DIM S$(15),A(3,5)
0030 INPUT "SUSPECT'S NAME: ",S$
.
.
.
```

(Input data here and fill array A.)

```
0100 WRITE FILE (0),S$
0110 FOR I=0 TO 3
0120   FOR J=0 TO 5
0130     WRITE FILE (0),A(I,J)
0140   NEXT J
0150 NEXT I
0160 GOTO 0030
.
.
.
```


Licensed Material-Property of Data General Corporation

Examples

```
0010 DIM REC$(48)        :48 bytes + 2 bytes (status) = 50 byte records.
0020 OPEN FILE(2,0),"DATA"
0030 INPUT "WHAT RECORD DO YOU WANT TO REWRITE? ",R
0040 POSITION FILE(2,50*R)       :50 is record size, R is record you want.
0050 READ FILE (2),STAT%,REC$    :Read 2 bytes into STAT%, rest into REC$.
0060 PRINT REC$                  :Look at record.
0070 INPUT "REWRITE IT? YES(0),NO(1): ",A
0080 IF A THEN GOTO 0400         :If "no", go to 400.
0090 INPUT "TYPE NEW RECORD: ",REC$
0100 POSITION FILE(2,50*R)       :Go back to beginning of record R.
0110 WRITE FILE (2),STAT%,REC$   :Write new record, overwriting old one.
.
.
.
```

End of Chapter

Licensed Material-Property of Data General Corporation

Chapter 4
Files

File Concepts

In this section, we discuss the following topics:

- Filenames
- File Attributes
- Sequential Files
- Random Files
- Contiguous Files
- Subfiles
- Dynamic Record Allocation
- Index Files

File Setup In this section, we discuss the following topics:

- File Creation
- PARAM File
- INITFILE Utility
- File Characteristics Array
- OPEN Utility
- Creating Index Files

  * INITFILE
  * INDEXBLD utility
  * IBUILD utility
  * TBUILD utility
  * XBUILD utility

File Access

In this section, we discuss the following topics:

- Input/Output Variables
- File Access Keywords
- Sequential Access
- Direct (Random) Access
- Linked-Available-Record Access
- Block Input/Output
- ISAM (Index File Access)
- File Locking

INFOS files are completely different from RDOS/DOS type files and regular Business BASIC access, so they are handled separately in Chapter 9.

File Maintenance and Screen Maintenance Utilities create and access files using keyboard operations -- you do not write programs, you simply use the utilities. They are also discussed separately: Chapter 5 is File Maintenance, and Chapter 6 is Screen Maintenance. The EDIT utility also creates, edits, inputs, and outputs ASCII text files; see Chapter 7 for the EDIT utility.

File Concepts

A file is a collection of data you can use for input to a program or output from a program. That includes database files, index files, master and subfiles, core image files (for SAVEd programs), print image files and listing files (for LISTed programs and formatted output), table files (associated with file maintenance), user directories and disk partitions, temporary scratch files, and devices (as, for example, line printer, card reader, and keyboard).

Filenames

RDOS and DOS allow access to any file by its filename. Filenames can have from 1 to 10 alphanumeric characters and a 0, 1, or 2 character extension separated from the name by a period. You can only use the dollar sign ($) at the beginning of filenames denoting devices, like $LPT for the line printer, $CDR for the card reader, etc. Here are sample filenames:

ABC.	file named "ABC" with no extension. Can be anything except a device.
MYFILE.SR	file named "MYFILE" with an "SR" extension to indicate that it is a source file.
MYFILE.RB	file named "MYFILE" that is completely different from "MYFILE.SR" above. Use extensions to make files with the same name unique. The "RB" extension indicates that this file is a relocatable binary file.
XY235.SV	file named "XY235" indicated as a save file by using the "SV" extension. A save file is a binary file, for example, a BASIC program, that was SAVEd.

Certain extensions have special meanings to RDOS and DOS. To prevent confusion you should avoid using them, although your RDOS or DOS system does not restrict their use. Some of the file extensions currently in use and their meanings are listed in Table 4-1.

Licensed Material-Property of Data General Corporation

Table 4-1. File Extensions

RDOS/DOS/Business BASIC File Extensions	
File Extensions	**Meaning**
.SR	Assembly language or BASIC source file, usually with comments to document the assembly language or BASIC program.
.LS	A LIST file of a BASIC program in ASCII format.
.OL	RDOS/DOS overlay file or BASIC save file designed to be SWAPped to only.
.TM	Temporary file created by a program for its own use.
.SC	Scratch file normally used by editor programs.
.DR	RDOS/DOS user directory or subpartition.
.CM	Interprogram communications file or common area.
.DS	Documentation source file for comments to programs.
.ER	Error message file.
.SV	BASIC save file (core image) for SAVEd program.
.	(no extension) BASIC save file or data file.
Business BASIC File Extensions	
.TB	Table file, associated with the File Maintenance Utility.
.SL	Business BASIC subroutine source or listing file in ASCII format with comments.
.Sn	Screen file maintained by the SM utility, where n is a number indicating the terminal type.

Devices are files with preassigned names usually beginning with a dollar sign ($). For example,

$LPT the primary line printer
$LPT1 the secondary line printer
$TTI teletype input (keyboard)
$CDR card reader
MT0 magnetic tape unit 0 (no dollar sign)
DP0 Disk unit 0 (no dollar sign)

Disks and mag tape drives are special devices that allow multiple files per unit. You can refer to specific files within a unit by using the format unit:filename. Tape filenames indicate the number of the file's relative position (0,1,2...) from the beginning of the tape. For example,

MT0:0 first file on tape unit 0.
MT0:1 second file on tape unit 0.
MT1:5 sixth file on tape unit 1.

Disks are made up of partitions that may contain subpartitions which may in turn contain user directories (i.e., collections of files).

If you're in the primary partition only, references to files on disk use the format partition:directory:filename. The partition can be omitted if it is a disk already initialized in the system. The directory can also be omitted if the file you want is in the current directory in which you are working. For example:

DP1:TONY file TONY on disk unit 1.

DP1:NBASIC:TEST.TM file named TEST.TM is in user directory NBASIC on disk unit 1.

MYFILE.2 file named MYFILE.2 in the current user directory.

A disk file directory is merely an index which lists the files in a portion of disk file space. A directory describes where certain items can be found. The disk file directory also contains other information about each entry such as the file's attributes (described next), the time the file was created, etc.

Your directory contains the names of your files. These files are the only ones you can access. Your directory also contains link entries. Link entries are dummy filenames that point to real files in other directories. For example, all devices have link entries in your directory so that you can access them. You can only access a file in another directory if it has a link name in your directory.

Licensed Material-Property of Data General Corporation

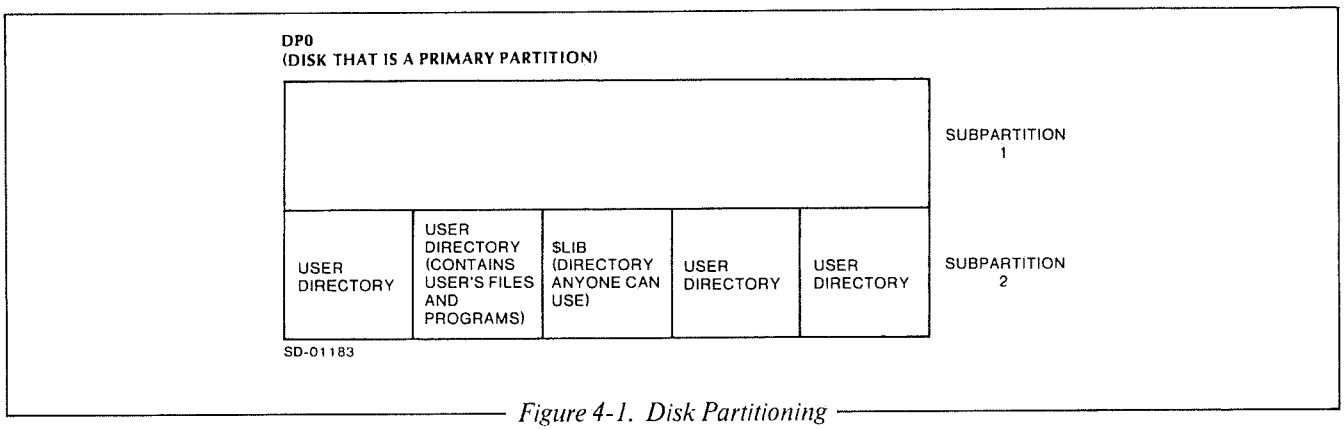


Figure 4-1. Disk Partitioning

File Attributes

Each file has its own particular characteristics, called attributes, defined in a directory along with its name. The BASIC CLI LIST command allows you to examine a file's attributes, size, and other information. You can change some attributes using the BASIC CLI commands, CHATR or CHLAT. Some attributes important to BASIC users are:

P Permanent file, cannot be deleted or renamed.
S Save file, core image load.
W Write protected, cannot be written to.
R Read protected, cannot be read from.
L Link entry, dummy filename allowing an alias reference to a file in another directory.

A Attribute protected, prevents further alteration of this file's attributes.

D Random file organization.
C Contiguous file organization.

NOTE: If a file has neither random (D) nor contiguous (C) attributes then it is organized sequentially. A discussion of all three types of file organization follows this section.

You can change some file attributes with the BASIC CLI CHATR command. Certain attributes cannot be changed, and any file with the A attribute cannot have its attributes changed. You can change link attributes with the BASIC CHLAT command. Both commands, CHATR and CHLAT, are listed in alphabetical order in Chapter 7, and if you want more information about attributes, look them up.

Sequential Files

Disk files without the C or D attribute, and all devices are sequentially organized. Information, in sequentially organized files, is stored in groups of disk blocks. The last word of each 256-word block stores a pointer to the next block in the file. If you are building a sequential file, the system simply appropriates the next available disk block when it needs storage, and constructs a pointer to the block; consequently, you don't have to worry about finding the necessary space or maintaining pointers. Space allocation is dynamic: sequential files appropriate space as they need it, so you can always append records to a sequential file, and records may vary in length.

Sequentially ordered blocks are sequential in this sense: after processing any given block, the system may step either to the previous block or to the next block in the series. If you want to access a record in the tenth block of a sequential file after accessing a record in the first block, the system would have to read the nine intervening blocks -- a time consuming process. For faster record access, you might want to use randomly organized files or contiguously organized files.

Random Files

Random file organization provides you with the best combination of flexible file structure and easily accessible data. Random files do not require that you read all intervening records when positioning to a specific record in the file. RDOS maintains a small sequential file of pointers to data blocks in the random file -- RDOS needs only to read this small file in order to find the location of the block data containing the desired record. It then accesses this block directly, resulting in faster access of records. Random files need slightly more disk space than sequential files due to the file of pointers that the system maintains. But, random files offer fast, direct access and dynamic allocation -- a random file allocates space as it needs it, and allows you to append records at will.

A record is the result of a single READ or WRITE operation, i.e., a record is actually a binary string of data, records may vary in length. However, by making your records fixed in length, you can have easier and faster access to both random and contiguous files. File access is discussed in detail later in this chapter.

Licensed Material-Property of Data General Corporation

Contiguous Files.

Contiguous files have a rigid structure though they provide the fastest access to data. Contiguous files are composed of a fixed number of disk blocks that are physically adjacent or contiguous on the disk. RDOS can easily calculate the location of a record and access it directly. These files can neither be expanded nor reduced in size, since by definition they occupy a fixed series of disk blocks. The disadvantage of using contiguous files is that you must allocate all the disk space they will ever require when you create the file. Their creation depends upon whether or not you have the required number of contiguous disk blocks available for your use. You can find out how many contiguous blocks you have in your directory by using the BASIC CLI DISK command, described in Chapter 8.

You can only process sequential files sequentially. You can process both random and contiguous files either randomly (directly) or sequentially. Normally you'd use sequential files for source input jobs -- files that contain data in ASCII format, and print image output files -- or files that are read/written all at once. Permanent database files are usually contiguous for faster access, and temporary files and files requiring dynamic allocation are usually random.

You can position to any file using the POSITION FILE statement, though the POSFL.SL subroutine supplied with Business BASIC gets you there faster and is especially recommended for locating subfiles. For dynamically allocated files, the subroutine GETREC.SL positions you to the next available record to which you can WRITE. Use the statements READ FILE, WRITE FILE, BLOCK READ FILE, and BLOCK WRITE FILE to access any data files, and use PRINT FILE and INPUT FILE to access any data in ASCII format. You'll find all of these statements in Chapter 3, and the provided subroutines are described in Chapter 7.

Subfiles

Business BASIC has a subfile access method. Subfiles are logical files that exist within RDOS physical files. Subfile capability fulfills the need for having more than 16 channels available in a program. The BASIC OPEN FILE statement (described in Chapter 3) only allows you to OPEN up to 16 physical files (channels 0 to 15) in one program. Since, however, you can subdivide a physical file into subfiles, the possibilities are endless -- you can open as many logical files as you want.

First, take a look at the OPEN FILE statement (Chapter 3). If you open a file that a program will access, you must supply a channel number for the file, and use the channel number to reference the file in sebsequent file access statements. You are only allowed 16 channels in one BASIC program -- channels 0 through 15. That might be enough for you, but if you want to access more files in larger application programs, you'd set up subfiles.

To set up and use subfiles, you need a PARAM file to hold file information (see next section), and you have to create a special array in your program called the file characteristics or C1 array. "C" is the variable you use to reference the array in subroutines and the OPEN program. The C1 array is a two-dimensional array that contains information about all the logical files (physical files or subfiles) you wish to use in your program. You can code the array in your program, use the OPEN utility program or the FINDFILE.SL subroutine supplied with Business BASIC in conjunction with the PARAM file. We explain how you set up and access files further in this chapter.

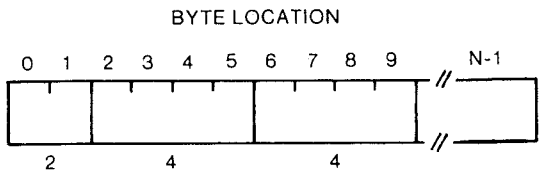
You can create the C1 array to use for all file access in your program, whether you use RDOS/DOS physical files only or Business BASIC subfiles built within physical files. With the C1 array, you can use the following subroutines: POSFL.SL to position to any record or any byte within a record; GETREC.SL to find the next available record to write to (for linked-available-records and dynamic record allocation; and DELREC.SL to delete records and make space available for new records. We describe how to use these subroutines in Chapter 7; if properly used, they will save you time and energy. Since these subroutines use the C1 array, you must create the C1 array to use them.

Dynamic Record Allocation

Dynamic record allocation, sometimes called "linked-available-record format" or "disk space management" means simply that a file can grow in size as it needs to when you write to it and it reuses space left by deleted records. Using the GETREC.SL and DELREC.SL subroutines with the C1 array, you can allocate files dynamically. Random files reuse space left by deleted records and allocate new space as needed. Contiguous files and subfiles can reuse space left by deleted records but cannot allocate any new space because they are fixed in length.

To have a file use dynamic record allocation, you need the following information in the file: the record number of the next available record (e.g., a record that was deleted), and the record number of the last record used. (e.g., the last record in the file, so that you can add on to the end of the file if no records were deleted). These record numbers must be maintained since they change with each file access. The subroutines, GETREC.SL and DELREC.SL, maintain record numbers for you, if you put them in record 0. Record 0, in a dynamic allocation file, is reserved to hold this information so that GETREC.SL and DELREC.SL can find it and maintain it for you. You can create your record 0 in your dynamically allocated file either by hard coding it, or by using the INITFILE utility (described in the next section and in Chapter 7).

Licensed Material-Property of Data General Corporation



where:

N = maximum number of bytes per record.

Byte locations:

0-1 = status flag (always -2 for record 0).

2-5 = record number of next available record (-1 if there are no deleted records).

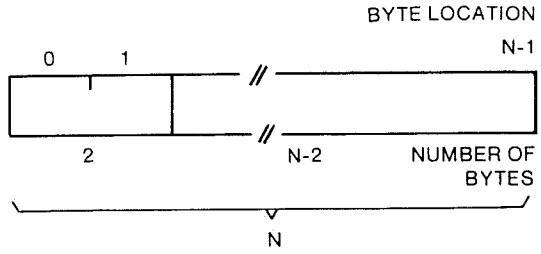
6-9 = record number of last record used - either last record in file or highest-numbered record written to.

N-1 = unused.
Record 0 is always the same size as the data records.

SD-01184

Figure 4-2. Record 0

Record 0 is shown in Figure 4-2. The record number of the next available record points either to the next record after the last record used, or to a deleted record chain. How does it point to a deleted record or to the last record used? The data records have status flags as shown in Figure 4-3.



where:

N = maximum number of bytes per record.

Byte locations:

0-1 = status flag >0 (equal to "record type" in File Maintenance files).

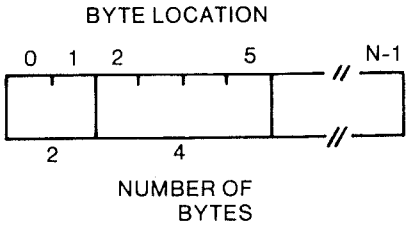
3 thru N-1 = data.

SD-01185

Figure 4-3. Data Record

Deleted records form a chain unless there is only one deleted record -- then the one deleted record is treated as if it were the last record in the deleted record chain. Figure 4-4 shows what they look like.

DELETED RECORD IN CHAIN



where:

N = maximum number of bytes per record.

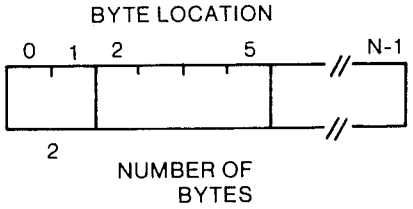
Byte locations:

0-1 = status flag is 0.

2-5 = pointer to next deleted record in chain (record number of next deleted record).

. . . N-1 = unused.

LAST RECORD IN CHAIN
OR
ONLY DELETED RECORD



where:

N = maximum number of bytes per record.

Byte locations:

0-1 = status flag is 0.

2-5 = pointer is -1 for no more deleted records.

. . .N-1 = unused.

SD-01186

Figure 4-4. Deleted Record Chain

Licensed Material-Property of Data General Corporation

The GETREC.SL and DELREC.SL subroutines look for the status flags and update record 0 accordingly. You don't have to bother with status flags (unless you want to do something special), but you do have to know they are there, and you can retrieve them. Status flags are two bytes long and you can retrieve or create them using a single precision (%) variable which only transfers two bytes in read/write operations. Whenever you READ a record from a dynamically allocated file, keep in mind that you will get this status flag along with the contents (unless you position to a byte within the record). Consequently, the first two bytes of an ordinary READ will most likely be a status flag, and you can either use it or not. Don't change status flags yourself unless you know what you're doing -- other system utilities use the status flags to tell if a record is active or deleted, and File Maintenance uses the specific values of status flags for active records.

The status flag of record 0 is always -2. The status flag of an actual data record must be greater than 0. If the file was created using the File Maintenance Utility (see Chapter 5), the status flag of a data record will equal the "record type," because you can have several types of records in one file which is maintained by File Maintenance. Status flags of deleted records must equal 0. Each deleted record has a pointer that points to the next deleted record in the deleted record chain. If the deleted record is the only or the last record in the chain -- then the pointer in the deleted record has a value of -1.

You can also access dynamically allocated files using the File Maintenance Utility (see Chapter 5). The INITFILE utility automatically sets up record 0 for you, and the GETREC.SL and DELREC.SL subroutines (with the C1 array) automatically maintain the pointers in record 0 for you. These routines are described in the "File Access" section of this chapter, as well as in Chapter 7. Read the next section, "File Setup", to find out how to use INITFILE, the C1 array, and how to create such files.

Index Files

Index files are dynamically allocated RDOS/DOS-type files, which can be random or contiguous files. They are conceptually no different from ordinary files, except that you use different routines to access them; and these routines provide an ISAM (Indexed Sequential Access Method) for your data files.

Index files point to data files. But index files are separate files with distinct names; they can be subfiles; and you can have an unlimited number of index files for one data file. Each index would hold a unique type of key. If, for example, you wanted to access a data file record using three different keys, like company name, corporate number, and corporate representative's name, you'd have three index files for that one data file: one for company name, one for corporate number, and one for corporate representative's name. You can also have duplicate keys in an index where two or more keys of the same value point to different records. You can also specify that you don't want any duplicate keys, and the index will do a search every time you add a key to make sure there isn't already a key with that value in the index.

Index files differ from data files because they are created in a special way. Index files are made up of 512-byte sectors (called "blocks") instead of records, and the access routines automatically determine how many index entries to put in one block. Block 0 of an index file (similar to record 0 of a dynamically allocated file) is the header block that contains information for the access routines. Block 0 will tell the access routines how many bytes to expect per entry (key and pointer), the number of keys per block (calculated by the INDEXCALC utility), the last available block (like the next available record), the internal block number of the highest level block, the blocking factor, and special bit flags. Block 0 is created automatically by the INITFILE utility in a question and answer dialogue. You can also create block 0 by using the INITINDEX.SL subroutine, the INDEXBLD utility, and the IBUILD, XBUILD and TBUILD utilities which create temporary indexes. We discuss index creating techniques in "File Setup".

The statements you can use with index files an ISAM type of access, allow automatic locking of the index file, and allow duplicate keys. These statements are: KADD, to add a key; KDEL, to delete a key; KFIND, to find a key; and KNEXT, to find the next key. Using KFIND with KNEXT, you can start at any key and read sequentially from that key; thus, using index files, you can read a data file in any order you choose without having to sort the file. These statements are discussed later in this chapter in "File Access," and each statement is also described in Chapter 3.

Licensed Material-Property of Data General Corporation

Business BASIC ISAM also allows "approximate look-ups," which means if you give a partial key to KFIND, it will find an equal or greater value of that key. This allows you to have one or more keys concatenated into one index, so that if you ask for a partial key, it will find an approximate match for it.

Block 0 of an index file should be created by using INITFILE, INITINDEX.SL, INDEXBLD, IBUILD, TBUILD, or XBUILD; you shouldn't try to build it yourself, but we show how it looks in Figure 4-5.

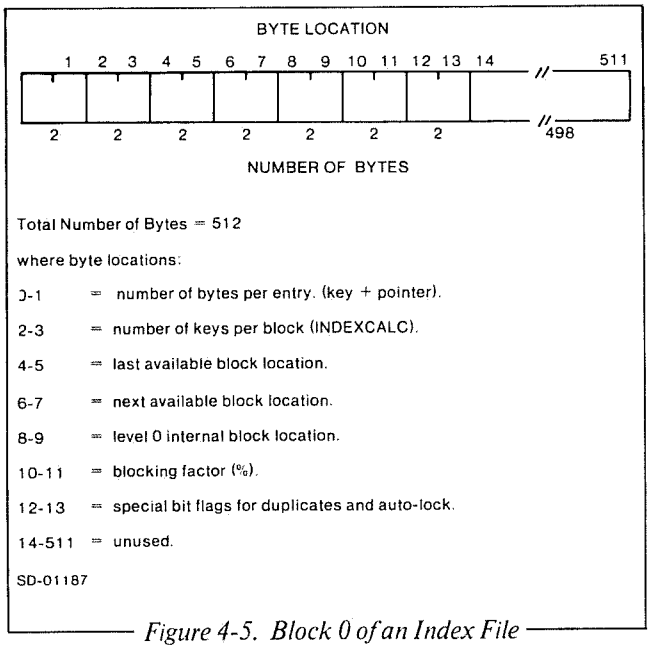

BYTE LOCATION
1 2 3 4 5 6 7 8 9 10 11 12 13 14
511
2 2 2 2 2 2 2
498
NUMBER OF BYTES
Total Number of Bytes = 512
where byte locations:
0-1 = number of bytes per entry. (key + pointer).
2-3 = number of keys per block (INDEXCALC).
4-5 = last available block location.
6-7 = next available block location.
8-9 = level 0 internal block location.
10-11 = blocking factor (%).
12-13 = special bit flags for duplicates and auto-lock.
14-511 = unused.
SD-01187

Figure 4-5. Block 0 of an Index File

Each block in an index file contains this information and individual key entries as shown in Figure 4-6.

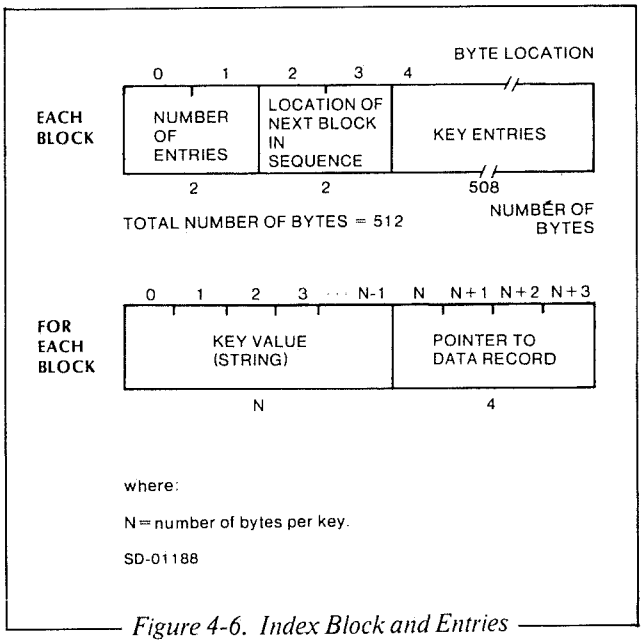


Figure 4-6. Index Block and Entries

There is always at least one key entry in any given index block. When space runs out in a particular index block (as you are adding keys), the block is subdivided into two new blocks using the blocking factor. You don't have to worry about blocking factors unless you are repacking an index to make it smaller. The blocking factor when you create an index file using KADDs is usually 75%.

The overall structure of an index file resembles a balanced multi-node tree, with the highest level block (called "Level 0") at the top. See Figure 4-7.

Licensed Material-Property of Data General Corporation

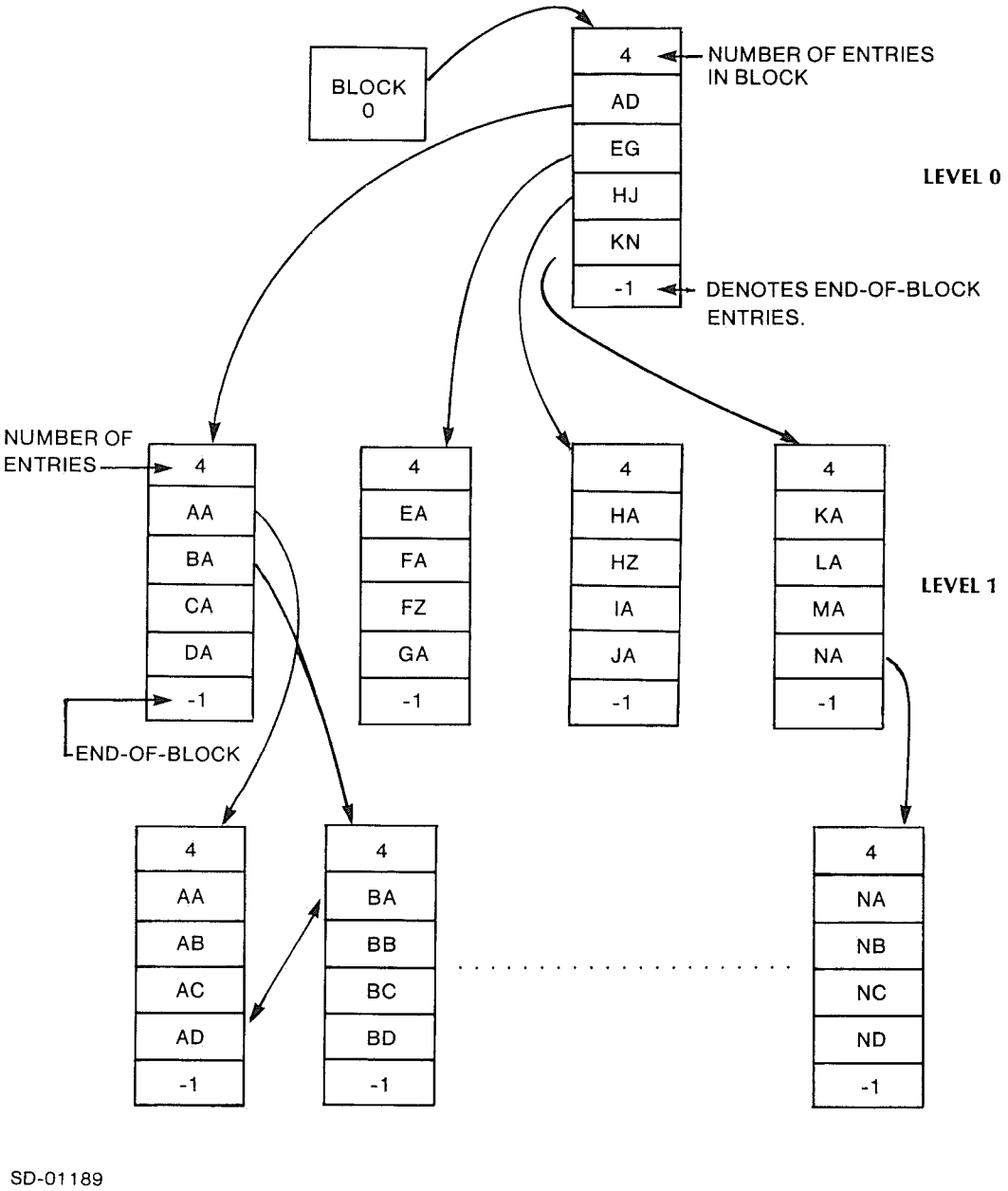


Figure 4-7. The Structure of an Index File

Licensed Material-Property of Data General Corporation

Block 0 in an index file points to the highest level (Level 0) block. The entries in Level 0 are searched until the first entry equal to or greater than the key you're looking for is found. The pointer associated with the entry found is either the location of the next block in the tree to be searched, or the pointer to the data record if the entry equals the key you were looking for. High level blocks point to lower level blocks by looking at the "next block in sequence" value in each block.

Index files are maintained in a sorted order. Thus you can read sequentially through a data file using any key or type of key as a starting point and read the index sequentially from that point. For example, if you want to read records in alphabetical order starting with "McMurphy", you would KFIND "McMurphy" and then KNEXT to the end of the index. You never have the time consuming problem of sorting a large data file. For more information about creating index files, see "File Setup". For more information about index file access, see "File Access".

File Setup

This section explains how to create logical files and index files and how you set them up for use in your programs. The File Maintenance Utility (see Chapter 5) also creates and sets up files for you to access directly from the keyboard. Since File Maintenance uses the techniques described in this section, to understand File Maintenance you should read this section as well as Chapter 5. See Chapter 9 for INFOS files.

File Creation

You can create sequential and random files directly in your program using OPEN FILE or the OPEN utility program. However, if you wanted them to be logical files (subfiles) and/or dynamically allocated, you'd have to do some tricky coding. Business BASIC provides an easier method for creating all types of files using INITFILE, and PARAM file, the file characteristics array C1, FINDFILE.SL, the OPEN utility, and the index building utilities, which are described in this section.

You must create contiguous files prior to using any of the aforementioned routines on them, because contiguous blocks must be allocated first. Create a contiguous file by using the CCONT command, a BASIC CLI command described in Chapter 8 which is similar to the RDOS/DOS CLI CCONT command.

Here is a summary of how to create files:

Sequential files-- Create sequential files using CREATE, and/or open them in your program with the OPEN FILE statement.

Random files-- Create random files using INITFILE to have dynamic allocation, or CRAND them and use File Maintenance, or create a temporary non-dynamic file using OPEN FILE, or simply CRAND them and worry about their contents later.

Contiguous files-- You must use CCONT to create contiguous files, then you can INITFILE them if you want, or use File Maintenance on them or just OPEN FILE them.

Subfiles-- You can create subfiles or logical files using CRAND, CCONT, or CREATE, but they *must* have an entry in the PARAM file. You can make a PARAM entry automatically using INITFILE, or you can put an entry in PARAM using File Maintenance.

Index files-- Index files are separate files that point to records in one or more data files. Create them using INITFILE. Alternatively, you can use the index building routines described later in this chapter if your data file is already set up to generate an index file, or use File Maintenance.

INITFILE and File Maintenance are provided to minimize the amount of coding you have to do. If you created random or contiguous files using only OPEN FILE or the OPEN program, you'd have to worry about PARAM entries and record 0 yourself, whereas INITFILE takes care of them for you.

PARAM file

The PARAM file contains file information that you need in order to have subfiles and/or in order to use the OPEN utility program. The PARAM file tells OPEN and other utility programs what subfiles and/or physical files are to be used with BASIC I/O routines. PARAM allows you to move and/or change the size of files without subsequent program modification -- all you have to do is update the record in the PARAM file that describes the file.

PARAM consists of records that are 42 bytes long. It is dynamically allocated (record 0 of PARAM describes the PARAM file itself), so you can add and delete as many records as you want while PARAM maintains its own space -- provided you use the proper routines to update PARAM. You can access PARAM either through File Maintenance (see Chapter 5) or through your own access routines (making use of GETREC.SL, DELREC.SL, and POSFL.SL). The OPEN utility program and the FINDFILE.SL subroutine access the PARAM file automatically, and INITFILE creates a PARAM record for you automatically.

You will find an empty PARAM file (with record 0) which comes with Business BASIC in the $LIB directory. You can set up individual PARAM files for each user directory, or set up links to the PARAM file in $LIB, whichever is more efficient to handle your needs.

You can write a new record to PARAM using conventional file I/O, or put information into a PARAM record using File Maintenance. The easiest way to create a PARAM record for a new file is to use the INITFILE utility (described in this chapter). INITFILE looks for your filename in PARAM, and if it's not found, INITFILE will ask you if you want to add it to PARAM. If you do, INITFILE asks you the appropriate questions.

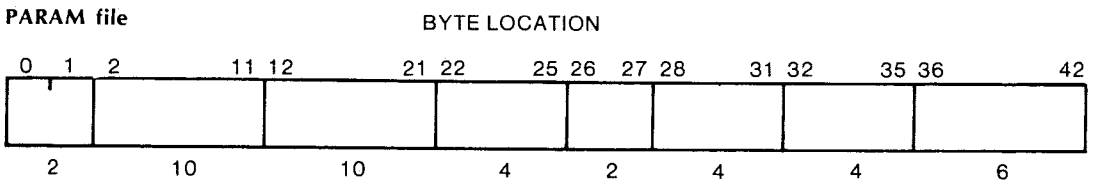
A PARAM file, including a table file for File Maintenance, should be provided. But if you run INITFILE and the following message comes back: PARAM FILE DOES NOT EXIST, DO YOU WANT TO CONTINUE?, it means that PARAM doesn't exist in your directory or something happened to the original one in $LIB. Since you need PARAM to have subfiles, and you would want PARAM in order to use some energy-saving utilities, you should know how to create a PARAM file. Here's how:

1. Create the PARAM file by doing a CRAND. CRAND is a BASIC CLI command discussed in Chapter 8.

2. Using File Maintenance, build record 0 of the PARAM file.

3. INITFILE all your data and index files to put the information in PARAM.

In the PARAM file, you create a record with file information as shown in Figure 4-8.



where byte locations:

0-1 = status flag is 1.

2-11 = subfile name (string) or logical filename

12-21 = filename of physical file that holds subfile, or filename of logical file if you are not using a subfile.

22-25 = byte pointer to start of subfile or logical file.

26-27 = record length of subfile or logical file.

28-31 = last record number of subfile or logical file.

32-35 = last record used (record only).

36-42 = unused.

SD-01190

Figure 4-8. File Information in PARAM File

Licensed Material-Property of Data General Corporation

INITFILE Utility

The INITFILE utility can be RUN, SWAPped to or CHAINed, or executed from the BASIC CLI by typing INITFILE. It asks you questions about your file and adds a record to PARAM if one doesn't already exist. The examples below provide a sample INITFILE dialogue. Example 4-1 shows a subfile within a file, Example 4-2 shows a logical file that is a physical file and not a subfile, and Example 4-3 shows a file that already has a PARAM entry. An example of an index file can be found later in this chapter under "Index Files".

Example 4-1

```
*RUN "INITFILE" )
INDEX(0), DATA(1) OR STOP(2)? 1 )
(You enter a 1 for a data file.)
LOGICAL FILENAME: BOVE )
(BOVE is the name of your logical file.)
FILE NOT IN PARAM FILE! DO YOU WISH TO ADD? YES )
(Yes, you want to make an entry for BOVE in PARAM.)
   (These questions only asked if you are making a PARAM entry. Answers are used to make the PARAM record
   for "Bove".)
   MASTER FILENAME: USER 1 )
   (BOVE resides in physical file USER1)
   BYTE OFFSET TO RECORD 0: 1500 )
   (BOVE's record 0 starts at byte location 1501 of USER1.)
   RECORD SIZE IN BYTES: 100 )
   (BOVE has records of maximum 100 bytes per record.)
   LAST RECORD NUMBER: 50 )
   (BOVE has a maximum of 50 records.)
SHOULD FILE BE NULL FILLED? NO )
(You would answer "YES" if the file once had data in it and you wanted to make sure that all the previous data is
erased before adding new data to the file.)
INDEX(0), DATA(1) OR STOP(2)? 2 )
(You can INITFILE another file if you want, or stop.)
*
```


Licensed Material-Property of Data General Corporation

In Example 4-2, the file, TONY, is a physical file, not a subfile, therefore the byte offset to record 0 is 0. This time we start out in the BASIC CLI (! is the BASIC CLI prompt):

Example 4-2

```
!INITFILE )

INDEX(0), data(1) OR STOP(2)? 1 )
LOGICAL FILENAME: TONY )
FILE NOT IN PARAM FILE! DO YOU WISH TO ADD? YES )
MASTER FILENAME: TONY )

(TONY is both the logical file and the physical file.)

   BYTE OFFSET TO RECORD 0: 0 )

   (TONY starts at the beginning of TONY.)

   RECORD SIZE IN BYTES: 128 )

   LAST RECORD NUMBER: 100 )
SHOULD FILE BE NULL FILLED? YES )
INDEX(0), DATA(1), OR STOP(2)? 2 )
!
```

Example 4-3 shows INITFILE SWAPped to from a program in order to initialize a subfile for access. The subfile already has a PARAM entry, so the MASTER FILENAME, BYTE OFFSET, RECORD SIZE, and LAST RECORD questions are not asked. You can force a new entry to PARAM by deleting the old entry using File Maintenance or DELREC.SL, and providing the new entry using File Maintenance or INITFILE. In this example we kept the old PARAM entry but filled the file with nulls because we intend to write an entire new file:

Example 4-3

```
.
.
0600 SWAP "INITFILE"
.
.
*RUN )
.
. (Program runs, then swaps to INITFILE.)
INDEX(0), DATA(1) or STOP(2)? 1 )
LOGICAL FILENAME: TONY)
SHOULD FILE BE NULL FILLED? YES )
INDEX(0), DATA(1), OR STOP(2)? 2 )
.
. (Program continues)
```

File Characteristics Array (C1)

The file characteristics array, C1, makes life easier for you if you're using subfiles and/or physical files and you want to use the file access routines GETREC.SL, DELREC.SL, and POSFL.SL. A subroutine called FINDFILE.SL will create the array for you, but you still have to DIMension the array. You can also use the OPEN utility program to supply the information for the C1 array *and* open the files for you; you usually use OPEN to open your files (instead of using the OPEN FILE statement) *or* you can use FINDFILE.SL to fill your C1 array without opening the files.

The C1 array contains the channel number of each subfile on which the physical file will be opened (you can have many subfiles opened on one physical file's channel), the byte offset to record 0 of the subfile or logical file, the file size (number of last record in the file), and the record size in bytes. The file access subroutines use this information to position you to the right place.

Licensed Material-Property of Data General Corporation

First, DIMension the C1 array (it must be called "C1.") as a two-dimensional array with *n-1* rows and 3 columns, where *n* is the number of logical files (subfiles and/or physical files) you want to use in the program. Since arrays begin at 0 and not 1, if you had a C1 array with 3 rows, you'd be able to open 4 files: file 0, file 1, file 2, and file 3. The column dimension corresponds to the following:

0 Channel number of the physical file containing this subfile, or channel number of the file if file is not a subfile; return by the OPEN utility or set by yourself. It is the channel number you associate with the file for all file access (see OPEN FILE statement in Chapter 3 for more information about channel numbers).

1 Byte offset to record 0 of subfile, i.e., subfile starting location in the physical file, or 0 if file is not a subfile.

2 File size, i.e., number of last record in the logical file, whether it is a subfile or a physical file.

3 Record size, i.e., bytes per record of logical file, whether it is a subfile or a physical file.

Figure 4-9 shows a picture of a C1 array and the program that filled it, using LET statements. C1 contains three files -- the first two are subfiles and the last one is a physical file.

		CHANNEL NUMBER	BYTE OFFSET	FILE SIZE	RECORD SIZE
		0	1	2	3
SUB1 (subfile of MASTER)	0	2	51200	1200	48
SUB2 (subfile of MASTER)	1	2	0	400	128
PHYS (physical file)	2	3	0	100	50

SD-01191

```
0010 DIM C1(2,3)                  :Dimension C1 to be a 3*4 array.
0020 LET C1(0,0)=2                :File 0 (SUB1) opened on channel 2,
0030 LET C1(0,1)=51200            :and begins at byte 51201 (block 100),
0040 LET C1(0,2)=1200             :and contains a max of 1200 records,
0050 LET C1(0,3)=48               :with 48 bytes in each record.
0060 LET C1(1,0)=2                :File 1 (SUB2) also opened on channel 2,
                                  :because SUB1 and SUB2 are in same physical
                                  :file MASTER, opened on channel 2.
0070 LET C1(1,1)=0                :SUB2 begins at byte 1 of MASTER,
0080 LET C1(1,2)=400              :contains 400 records
0090 LET C1(1,3)=128              :with 128 bytes in each record.
0100 LET C1(2,0)=3                :File 2 (PHYS) opened on channel 3,
0110 LET C1(2,1)=0                :starts at byte 1 since it is a physical file,
0120 LET C1(2,2)=100              :and contains a max of 100 records,
0130 LET C1(2,3)=50               :with 50 bytes in each record.
0140 REM -- NOW WE OPEN OUR FILES
0150 LET C%=C1(0,0)               :Channel number now in C%.
0160 OPEN FILE(C%,0),"MASTER"     :This will allow access to both SUB1 and SUB2.
0170 LET C2%=C1(2,0)              :Channel number of PHYS now in C2%.
0180 OPEN FILE(C2%,0),"PHYS"
.
.
.
```

Figure 4-9. C1 Array

Licensed Material-Property of Data General Corporation

In the example program in Figure 4-9 you can change the record size, channel number, or file size of any of the files; all you'd have to do is change the values in the C1 array. Now that we have a C1 array in our program, we can use the subroutines GETREC.SL to get an available record to write to, POSFL.SL to position to any record in the logical (subfile or physical) file, and DELREC.SL to delete any record in the logical file.

FINDFILE.SL is a subroutine that builds the C1 array for you, except that it leaves column 0 (channel number) blank and returns the next available channel number to you in a variable (C%). FINDFILE.SL is described in Chapter 7, and its entry point is 7800 (i.e., GOSUB 7800). Use FINDFILE.SL instead of the OPEN utility to build your C1 array without opening your files. FINDFILE.SL gets the information from the PARAM file. If a PARAM entry doesn't exist, it asks you for the byte offset, record size, and file size, but it does "not" create the PARAM entry for you.

First you supply FINDFILE.SL with the subfile name and the logical file number. Create a subfile name in variable X$. Your logical file number corresponds to the row in the C1 array (e.g., logical file number 2 is put in row 2 of C1, logical file number 0 is put in row 0 of C1, etc.), and you assign your logical file number to variable F%. You can then GOSUB to FINDFILE.SL at line 7800 (don't forget to ENTER "FINDFILE.SL") and when the subroutine RETURNs, you will have a C1 array with the 0 column blank, a variable X$ with the physical filename for the logical file (subfile or physical file), and a variable C% with the next available channel number you can use with an OPEN FILE statement. Example 4-4 uses FINDFILE.SL to fill the C1 array with the same values used in Figure 4-9.

Example 4-4

```
0010 DIM C1(2,3)                    :Dimension C1 to be a 3*4 array.
0020 DIM X$(10)                     :Dimension X$ to max filename size.
0030 LET X$="SUB1"                  :SUB1 is a subfile in MASTER,
0040 LET F%=0                       :and is logical file 0.
0050 GOSUB 7800                     :Go to FINDFILE.SL subroutine.
0060 GOSUB 0200                     :Go to verification routine.
0070 LET X$="SUB2"                  :SUB2 is in MASTER,
0080 LET F%=1                       :and is logical file 1.
0090 GOSUB 7800                     :Go to FINDFILE.SL subroutine.
0100 GOSUB 0200                     :Go to verification routine.
0110 LET X$="PHYS"                  :PHYS is a physical file,
0120 LET F%=2                       :and is logical file 2.
0130 GOSUB 7800                     :Go to FINDFILE.SL subroutine.
0140 GOSUB 0200                     :Go to verification routine.
0150 STOP
0200 REM -- VERIFICATION ROUTINE
0210 PRINT X$                       :Prints name of physical file associated of subfile.
0220 LET C%=C1(F%,0)                :Assign channel number to 0 column.
0230 PRINT C1(F%,0),C1(F%,1),C1(F%,2),C1(F%,3)
0240 RETURN


*ENTER "FINDFILE.SL" )
*RUN )
MASTER
2     51200     1200     48
MASTER
2     0         400      128
PHYS
3     0         100      50
STOP AT 0150
*
```


Licensed Material-Property of Data General Corporation

When you use FINDFILE.SL or the OPEN utility program, each finds an available channel number for your physical file (any subfile that exists in a physical file must use the channel number of the physical file, i.e., the physical file must be opened in order to access any subfile in it). You can, alternatively, open your physical file in an OPEN FILE statement and assign a channel to it yourself. But this is inconvenient if you're doing a lot of file access, since you might have to find out what channels are still available to you by using the UCHANS utility program to display user channel assignments (described in Chapter 7).

The open utility program finds a channel for you and supplies the information you need for the C1 array. It opens the physical file for your subfile, but you need a PARAM file entry for your subfile. FINDFILE.SL also finds a channel number for your physical file, but it does not open the file for you -- you must explicitly open it yourself. FINDFILE.SL will also work without a PARAM entry for the file. When you execute it without a PARAM entry it will ask you the same questions INITFILE asks you, but it will not make a PARAM entry for you. You should avoid confusion by making all your files consistent: start with a PARAM entry (using INITFILE) and then use the prewritten routines to create the C1 array and open your files.

We suggest methods which use our subroutines and utilities that are written in BASIC. If you wanted to use your own unique method to handle unique cases, or if you wanted to modify our routines to handle your specific applications, you might want to look at a listing of these subroutines and utilities since they are written in BASIC and can be easily changed. All subroutines supplied with Business BASIC are in ASCII format and you can ENTER, LIST, and modify them (all subroutines have the .SL extension). All utilities can be LOADed and then LISTed. If you modify a subroutine or utility, assign a new filename to the modified program. You should LIST new subroutines to a new ASCII file, and SAVE new utilities under a new name (do not REPLACE the old one).

OPEN Utility

The OPEN utility (also described in Chapter 7) finds an available channel for your physical file (physical files can contain many subfiles, all opened on the same physical file channel), opens your physical file, and returns the information for your C1 array. You need to have a PARAM entry for your subfile or physical file.

The OPEN utility provides the easiest way to open any file in a program. If the record size changes, you only have to change the PARAM entry for the record and one line of your program -- the line that passes information to the OPEN utility. You pass information to the OPEN utility in a string variable, X$, by passing X$ to the common area with a BLOCK WRITE statement.

First, dimension your X$ string to 512 bytes. You then assign a string literal to X$ that contains the logical filename and the mode of each file you want to open. The logical filename is the filename entry in PARAM -- it can be a subfile or a physical file. There is no limit to the number of subfiles you can open, but since you can only fit 512 bytes of information in X$, if you needed more, you'd have to set up another X$ and use OPEN again.

Here is an example of the string you need to supply to OPEN:

```
0020 LET X$="SUB1,5,SUB2,5,PHYS,6",FILL$(0)
```

There are three files in this string: SUB1, SUB2, and PHYS. Whether they are subfiles or physical files doesn't matter -- the information about them exists in PARAM, and OPEN will know what to do with them. Notice that we fill the string with nulls to the end -- FILL$(0). You must fill the entire string to its dimensioned length or an error will occur when you BLOCK WRITE the string. No imbedded spaces are allowed. After building the string X$, you send it into the common area with this statement:

```
0030 BLOCK WRITE X$
```

With X$ in the common area, you can SWAP to OPEN, and OPEN will gather the information in the common area and use it to open your files.

Notice that we used the numbers 5, 5, and 6 respectively for each file. These are *mode* numbers. Whenever you open a file, you must specify a mode. All the modes are discussed with the OPEN FILE statement in Chapter 3, but Table 4-2 is a quick summary:

Table 4-2. OPEN Modes

Mode	Type of Open	Type of I/O	Type of Access	Rule
0	Exclusive	Input/output	Random	Create file if it doesn't exist
1	Exclusive	Output only	Sequential	Delete, then create file.
2	Exclusive	Output only	Sequential	Append to existing file or create file.
3	Shared	Input only	Sequential	File must exist.
4	Shared	Input only	Random	File must exist.
5	Shared	Input/output	Random	File must exist.
6	Exclusive	Input/output	Random	File must exist.
7	Exclusive use of mag or cassette tape	Input/output	MTDIO access	Tape must exist; tape file may be created, deleted, or appended to.

Exclusive as opposed to *shared* means that you're the only user with access to that file at that time. Only if you open a file in a mode that is *shared* can another user also open that file at that same time. This has nothing to do with the LOCK/UNLOCK statements which are discussed separately in the next section of this chapter. You can only have random access to random or contiguous files. Random access is the ability to position to and read or write a record directly without having to read or write intervening records. Sequential access -- reading or writing from beginning to end -- can occur with any type of file except an index file. Mode 7 (direct mag tape I/O) is a special case where you can only use MTDIO statements.

Be careful when specifying modes, particularly when you are opening more than one subfile in a single physical file. One physical file is opened for all the subfiles in it, but if you tried to open a subfile in mode 5 and another subfile in the same physical file in mode 6, a conflict would occur: you can't open a physical file as both shared and exclusive. Consequently, the OPEN program checks this and, when it receives requests for multiple modes in the same physical file, it chooses the least severe mode in which to open the file. Therefore, if SUB1 and SUB2 were subfiles in PHYS, and we tried to open SUB1 and SUB2 in mode 5 and PHYS in mode 6, the OPEN program would open PHYS in mode 5 as shared instead of mode 6 (exclusive).

Let's look again at our string passing statements:

```
0010 DIM X$(512),C1(2,3)
0020 LET X$="SUB1,5,SUB2,5,PHYS,6",FILL$(0)
0030 BLOCK WRITE X$
0040 SWAP "OPEN"
```

X$ is now in the common area, and the OPEN utility program is SWAPped to. OPEN reads the common area, finds SUB1, SUB2, and PHYS entries in PARAM (or returns an error if not found in PARAM), figures out which physical files contain these subfiles (if they are subfiles), looks for an available channel for each physical file it has to open, and puts all the information needed to fill the C1 array back into the X$ string. In our example, both SUB1 and SUB2 exist in physical file MASTER, and PHYS is a separate physical file. Therefore, the OPEN program opens MASTER and PHYS on two separate channels, and returns the same channel number for SUB1 and SUB2 since they are both opened on the same channel when OPEN opens MASTER. You never have to worry about the physical filenames of subfiles, their byte offset, their record and file sizes, their channel numbers, as long as you set up your C1 array.

When OPEN finishes executing, control returns the program to the next line after the SWAP line, line 50.

Licensed Material-Property of Data General Corporation

```
0010 DIM X$(512),C1(2,3)
0020 LET X$="SUB1,5,SUB2,5,PHYS,6",FILL$(0)
0030 BLOCK WRITE X$
0040 SWAP "OPEN"
0050 BLOCK READ X$
```

At this moment, X$, retrieved from the common area witha BLOCK READ, is a binary string of data that needs to be converted into a numeric array. You have to use the ASC function with a byte pointer in the string to extract the elements from the string. Here is a simple way of doing this:

```
0050 BLOCK READ X$
0060 LET K=1
0070 FOR I=0 TO 2
0080   FOR J=0 TO 3
0090     LET C1(I,J)=ASC(X$(K,K+3))
0100     LET K=K+4
0110   NEXT J
0120 NEXT I
```

K is the pointer to the first element in the string. For each logical file, file 0, file 1, and file 2, and for each dimension of the C1 array, we extract the element and put it in the C1 array, then we bump the string element pointer.

This routine fills the C1 array for you. Notice that three logical files were specified: SUB1, SUB2, and PHYS, and that C1 is dimensioned to 3 rows corresponding to 3 logical files: file 0, file 1, and file 2. If you use these routines, the order in which you place the filenames in X$ determines the order in which they are assigned to the C1 array; e.g., SUB1 will be file 0 in row 0, SUB2 will be file 1 in row 1, and PHYS will be file 2 in row 2.

With the C1 array filled and the files opened, you can now position to a record in the file using POSFL.SL, or find an available record to write to using GETREC.SL, by setting the variable F% equal to the logical file number. In example 4-5 we use POSFL.SL to position to record 14 in the file, and we use the variable C%, returned by POSFL.SL, for the channel number argument in the READ FILE statement. POSFL.SL creates C% using information from the C1 array. The next section of this chapter, "File Access," describes these routines further.

Example 4-5

```
0010 DIM X$(512),C1(2,3),REC$(48)       :Record size of SUB2 is 48.
0020 LET X$="SUB1,5,SUB2,5,PHYS,6",FILL$(0)
0030 BLOCK WRITE X$                     :Send logical file info into common.
0040 SWAP "OPEN"                        :OPEN will return X$ with C1 array.
0050 BLOCK READ X$                      :Retrieve info from common area.
0060 LET K=1                            :Pointer to first element in string.
0070 FOR I=0 TO 2                       :For each logical file, 0, 1 and 2,
0080   FOR J=0 TO 3                     :and for each dimension of C1 array,
0090     LET C1(I,J)=ASC(X$(K,K+3))     :extract element, put in C1 array.
0100     LET K=K+4                      :Bump pointer 4 bytes.
0110   NEXT J
0120 NEXT I
0130 REM -- POSITION TO AND READ ANY RECORD IN FILE SUB2
0140 INPUT "TYPE RECORD NUMBER: ",R1    :R1 variable used by POSFL.SL.
0150 LET F%=1                           :F% used by POSFL.SL for logical file.
0160 GOSUB 9610                         :Position to record R1 in file F%
                                        :using POSFL.SL.
0170 READ FILE (C%),REC$                :C% is channel number of SUB2's
                                        :physical file, returned by POSFL.SL.
.
.
.
```


Licensed Material-Property of Data General Corporation

Creating Index Files

There are several utilities and routines that create an index file for you and maintain ISAM access. You can create an index from scratch, create it as you create a table file for a data file in File Maintenance, rebuild it from a former index, build it from a temporary index (tag file), or build it directly from a data file. Here is a quick summary of ways to create an index file; following the summary is a full description of each.

INITFILE (utility)

Initializes an index file the same way it initializes a data file. With a question and answer dialogue, it builds block 0 of the index file. You can then use KADDs to fill the index, or INDEXBLD, IBUILD, XBUILD, or File Maintenance while building the data file.

INITINDEX.SL (subroutine)

Initializes an index file by creating block 0 and block 1 from input arguments. Entry point is at 7700. The result is the same as INITFILE.

INDEXBLD (utility)

Builds an index from an input file. Input file can be another index (rebuild), a temporary index (tag file), or a data file. INDEXBLD calls TBUILD if input is a data file, and calls IBUILD to build index from a tag. It optionally calls XBUILD to build index from data file using KADDs, and is faster if data file is not sorted.

IBUILD (utility)

Builds an index from a sorted data file or tag file. Data file or tag file must be already opened and sorted in ascending order. IBUILD builds block 0 from input arguments. INDEXBLD calls this utility.

TBUILD (utility)

Builds a temporary index (tag file) from an index file or a data file. Tag file is sorted so that it can be input to IBUILD. INDEXBLD calls this utility.

XBUILD (utility)

Builds an index file from a sorted or unsorted data file using KADDs. Data and index files must already be opened; XBUILD builds block 0 using input arguments. INDEXBLD optionally calls this.

File Maintenance (utility)

Creates and builds up to 3 index files for one data file when you are setting up your table file for the data file and filling your data file using File Maintenance. Everything pertaining to File Maintenance is discussed in Chapter 5.

INITFILE

The INITFILE utility initializes an index file (as well as a data file) by building block 0 of the index file and making an entry for the index file in the PARAM file, if an entry doesn't already exist. You need block 0 in an index file or the Business BASIC ISAM to work (the KADD, KFIND, KDEL, and KNEXT statements use block 0). You need a PARAM entry for the index file if your index file is a subfile; otherwise, a PARAM entry is not necessary but it does make several routines run faster.

Index files can be subfiles, but it is not good practice to put index files and the data file they point to in the same physical file. Index files are created as random; if you want a contiguous index file, you have to CCONT the file first.

INITFILE can be RUN, SWAPped to, CHAINed, or executed from the BASIC CLI by typing INITFILE. Example 4-6 shows INITFILE creating an index file and the PARAM entry for the index file.

Licensed Material-Property of Data General Corporation

Example 4-6

*RUN "INITFILE")

INDEX(0), DATA(1) OR STOP(2)? 0)

(Enter a 0 for an index file.)

LOGICAL FILENAME: TBINDEX)

(TBINDEX is logical filename for index file.)

FILE NOT IN PARAM FILE! DO YOU WISH TO ADD? YES)

(The followint questions are asked to make an entry in PARAM.)

MASTER FILENAME: TBMASTER)

(TBMASTER contains subfile TBINDEX.)

BYTE OFFSET TO RECORD 0: 2000)

(TBINDEX starts at byte 2001 of TBMASTER.)
(The question "Record Size in Bytes" is not asked, because index files always use blocks, sometimes called "sectors," as records with 512 bytes in each block.)

LAST RECORD NUMBER: 8)

(8 is the number of 512-byte blocks needed in the index file. You calculate this by running the utility INDEXCALC, described after this example.)

SHOULD FILE BE NULL FILLED? NO)

(You would answer "YES" if the index file once had entries in it and you wanted to make sure that all previous entries are erased before adding new entries to the file.)

BYTES PER KEY: 10)

(The length of a key entry is 10 bytes. This does not include the pointer (4 bytes) to the data record. The additional 4 bytes for pointer is automatically added.)

BLOCKING FACTOR (%PERCENT): 50)

(When adding keys to an index, a block that is overfilled subdivides exactly in half, which is a 50% blocking factor. If you are using KADDs to build an index, 50% is always used. You can change the blocking factor if you use IBUILD of INDEXBLD to rebuild your index. A greater or lesser density can be specified; a high blocking factor will rebuild the index with more density, i.e., smaller.)

DUPLICATE KEYS ALLOWED? YES)

(Block 0 of index files contains a flag to indicate whether duplicate keys are allowed for this particular index. A duplicate key is a key with the same value as another key but it points to a different record in the data file. The answer sets this flag to allow duplicate keys in TBINDEX.)

INDEX(0), DATA(1) OR STOP(2)? 2)
*

Licensed Material-Property of Data General Corporation

INITFILE builds block 0 of TBINDEX and writes a dummy entry into the first block of the index. All keys added to the index will be inserted before this high-value dummy entry that marks the last key in the index.

In example 4-6, we answered the question LAST RECORD NUMBER with the value 8. We got this value by running INDEXCALC. INDEXCALC (described also in Chapter 7) is a utility that will calculate the number of sectors (512 byte blocks) needed for an index, given the size of the key entry, the size of the data record that holds the key (in the data file), the maximum number of records in the data file, and the index blocking factor. Example 4-7 is a sample run of INDEXCALC:

Example 4-7

```
*RUN "INDEXCALC" )
BYTES PER KEY: 10 )

(TBINDEX's keys are 10 bytes long.)

BYTES PER RECORD: 100 )

(The data file TBINDEX points to has 100 bytes per
record.)

MAXIMUM NUMBER OF RECORDS: 100 )

(The data file TBINDEX points to has a maximum of
100 records.)

INDEX BLOCKING FACTOR: 50 )

(When creating an index, use a blocking factor of 50 (in
case you need to make entries in the future). When
rebuilding an index, specify whatever blocking factor
you want -- see INDEXBLD.)

INDEXCALC returns this information:

36 KEYS PER INDEX BLOCK
6 BLOCKS AT LEVEL 0
1 BLOCKS AT LEVEL 1
8 SECTORS IN INDEX

(Use this value for the LAST RECORD NUMBER
question in INITFILE.)

20 SECTORS IN FILE

BYTES PER KEY: ikey (interrupt key)
*
```

INITINDEX.SL is a subroutine that performs a complete INITFILE routine for an index based on input arguments. See Chapter 7 for a description of INITINDEX.SL. After you use INITFILE to create and initialize your index file, you can build the index using KADDs, INDEXBLD, IBUILD, or XBUILD. If you are creating and filling a data file, you should use KADD in the same program to also fill your index file. Both files are opened separately (if they have PARAM entries, you can use the OPEN utility for both), and you would WRITE the record and then KADD the key for the record to the index file.

INDEXBLD

INDEXBLD is a utility program that allows you to build an index file from a data file, to build an index file from a tag file, and to rebuild an index file. INDEXBLD builds an index from a file you input to it. You can create the input file as a dynamically allocated and filled data file, as a sorted tag file, or another index file.

If the input file is a dynamically allocated data file, INDEXBLD calls TBUILD to build a sorted tag file for the data file, and then it calls either IBUILD or XBUILD (your choice) to build the index from the sorted tag file. IBUILD runs fast if the data file was previously sorted; otherwise, XBUILD (using KADDs) is faster.

If the input file is a sorted tag file built by TBUILD (see the following description of TBUILD, or refer to Chapter 7's description of TBUILD, for more information about tag files), INDEXBLD calls IBUILD or XBUILD bypassing the TBUILD routine to build a sorted tag file.

If the input file is another index file, a rebuild is assumed, and you can specify a new blocking factor if you want. The index is written sequentially into a scratch file, a new index is initialized (building block 0), and then rebuilt using the specified blocking factor. A high blocking factor (e.g., 90%) means the index is packed with a higher density and takes up less space; however, we do not recommend that you use a high blocking factor for highly volatile files. We recommend a blocking factor of 50% for all new index files.

Licensed Material-Property of Data General Corporation

Example 4-8

*RUN "INDEXBLD")

(INDEXBLD looks for a PARAM file; if missing, it issues a warning message.)

INPUT FILENAME: CUST)

(Input file's logical filename is CUST.)

INPUT FILE IS INDEX(0), DATA(1), OR SORTED TAG(2): 1)

(CUST is a data file.)

USE IBUILD(0) OR KADD(1): 1)

(INDEXBLD calls XBUILD instead of IBUILD. XBUILD is faster (using KADDs) for unsorted data files. IBUILD is better if you want to specify a blocking factor. This question only asked if input is a data file.)

(The PARAM file is now checked to see if the logical input file has an entry. If PARAM or the entry for the input file is missing, a warning message is issued, and the following questions are asked, but a PARAM entry is *not* made.)

BYTE OFFSET TO FIRST RECORD: 0)
RECORD SIZE IN BYTES: 100)

(Data file CUST has records of 100 bytes each.)

MAXIMUM NUMBER OF INPUT RECORDS: 100)

(Data file CUST has 100 records.)

CHECK FOR DELETED RECORDS; YES(1),NO(0): 1)

(This question only asked if input file is a data file. Remember, only data files that are dynamically allocated can be input to INDEXBLD. If "YES", deleted records (first two bytes -- status -- of a deleted record is 0) will be omitted from the tag file INDEXBLD creates. The next 3 questions are asked only if input file is a data file.)

NUMBER OF FIELDS IN KEY: 2)

(We specify 2 fields in our key: customer number and customer name. Our customer number field will need 4 bytes, and our customer name will need 6 bytes. Our customer name field is actually a crammed field -- where every 3 bytes are crammed into 2 bytes -- so when a key of 9 bytes is added to index, it only takes up 6 bytes in the index.)

Licensed Material-Property of Data General Corporation

Example 4-8 (continued)

(The next two questions are asked for each key field; since we specified 2, the questions are asked twice.)

KEY FIELD 1 LOCATION IN DATA RECORD (BYTE 0-N): 0)

(Key field 1 starts at byte 0 of data record.)

KEY FIELD 1 FIELD LENGTH IN BYTES: 4)

(Key field 1 is 4 bytes long, contains customer number.)

KEY FIELD 2 LOCATION IN DATA RECORD (BYTE 0-N): 4)

(Key field 2 starts at byte 4 of data record.)

KEY FIELD 2 FIELD LENGTH IN BYTES: 6)

(Key field 2, customer name, is 6 bytes long. No padding of individual fields occurs, but the total key length is padded to make the length an even number of bytes if necessary.)

INDEX FILENAME: CUSTINDEX)

(CUSTINDEX is the logical name of the new index file to be created. INDEXBLD looks in PARAM for an entry; if not found, INDEXBLD asks the following question but does *not* make a PARAM entry for you.)

BYTE OFFSET TO FIRST RECORD: 0)

(CUSTINDEX starts at byte 0 of its physical file.)

(The following question is not asked if you used XBUILD instead of IBUILD.)

BLOCKING FACTOR (%):

(Since we specified KADDs instead of IBUILD, this question was not asked, but we include it here for your information. If you specified IBUILD, you can now specify a new blocking factor: a high blocking factor, e.g., 90% will pack the index with a higher density, so that it will use less space. We recommend that you use 50% for volatile files so that you have no trouble rebuilding the index at a later date.)

DUPLICATE KEYS ALLOWED; YES(1),NO(0): 1)

(Yes, we allow duplicate keys, which are keys of the same value pointing to different data records, because several of our customers have multiple transactions. After this question, program runs to completion.)

*

For examples of INDEXBLD building an index from a tag file and rebuilding an index from another index file, see INDEXBLD in Chapter 7.

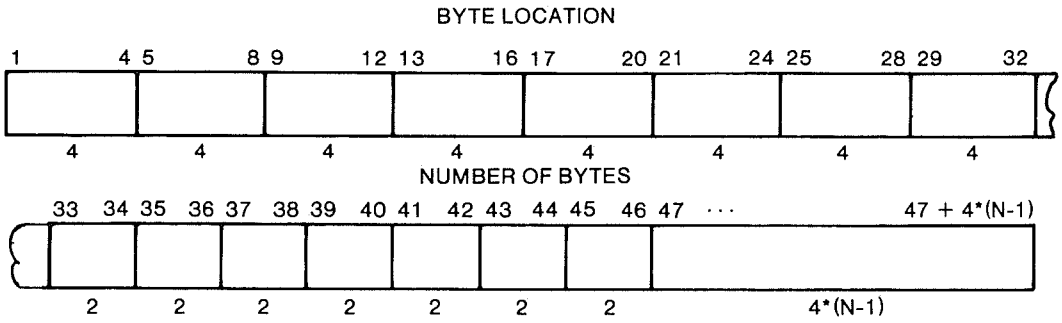
IBUILD

IBUILD builds an index file from a sorted data file or a sorted tag file (see TBUILD for sorted tag file information) with a specified blocking factor. IBUILD is fast, because the index levels are built in one pass, but your input file must be a data file or tag file sorted in ascending order. INDEXBLD builds a tag file and sorts it by calling QFILESORT (another utility program), so that you can specify IBUILD instead of KADD.

If the input file is a data file, the key may be composed of one or more individual fields, but you must specify the location of the key fields and their sizes. If the input file is a tag file, keys can also have multiple fields, but the location of the key is assumed to be first byte of the tag record. The record pointer in a tag file is assumed to be the 4 bytes following the key, whereas if a data file is input, IBUILD computes the record pointer when it retrieves the key from the data file.

To use IBUILD, you must build a string and send it into the common area with a BLOCK WRITE statement, and SWAP to IBUILD. See figure 4-10 to see what the string should look like.

Licensed Material-Property of Data General Corporation



where:

N = number of key fields.

Byte locations:

1-4	= channel number of input file.
5-8	= byte offset to record 0 of input file.
9-12	= maximum number of records in input file.
13-16	= number of bytes per record in input file.
17-20	= channel number of new index file.
21-24	= byte offset to record 0 of new index file.
25-28	= maximum number of records in index file (i.e., number of sectors).
29-32	= number of bytes per record in new index file, *always* 512.
33-34	= blocking factor, keys per block.
35-36	= flag to allow duplicate keys: 1=yes, 0=no.
37-38	= flag to check for deleted records: 1=check, 0=no check.
39-40	= total key length in bytes.
41-42	= number of key fields, or -1 if input is a tag file, -2 if input is another index file.
43-44	= starting location (relative to 1) of 1st key field. Not used for tag file or index file input.
45-46	ending location (relative to 1) of 1st key field.
47 thru 47+4*(N-1)	= starting and ending locations for each key field, not used if only 1 key field or if input file is a tag file or another index file.

SD-01192

Figure 4-10. IBUILD String

Licensed Material-Property of Data General Corporation

As you can see, you need to know the channel number of the input file and the new index file, which means they must be created and opened for use before you use IBUILD. If you use a C1 array in your program, you can plug information from the C1 array into this string. The number of bytes per record of the new index file is always 512 (512-byte sectors, called blocks). Do *not* express the blocking factor as a percentile, but as the number of keys per block. You calculate that by running INDEXCALC. You also determine the number of sectors in the new index file (maximum number of records) by running INDEXCALC. The number of key fields is irrelevant if the input file is a tag file or another index file, as well as the starting and ending locations of the fields. You must specify the starting and ending locations of each key field in the data record if your input is a data file. These locations assume that byte 1 is the first byte (not 0). Figure 4-9 shows a sample program that uses IBUILD.

Example 4-9

```
0010 DIM X$(512)
.               :Code to open both files goes here.  Channel number for input
.               :file is C%, channel number of new index file is CI%.
.
0600 LET X$=CHR$(C%,4),CHR$(0,4),CHR$(100,4),CHR$(100,4),CHR$(CI%,4)
0610 LET X$(0)=CHR$(0,4),CHR$(8,4),CHR$(512,4),CHR$(36,2),CHR$(1,2)
0620 LET X$(0)=CHR$(1,2),CHR$(10,2),CHR$(2,2),CHR$(1,2),CHR$(4,2)
0630 LET X$(0)=CHR$(5,2),CHR$(10,2),FILL$(0)
0640 BLOCK WRITE X$
0650 SWAP "IBUILD"
0660 STMA 1,1,ERCODE            :Error code returned from IBUILD.
0670 IF ERCODE=0 THEN GOTO 0700         :If no error, proceed to 700.
.
.
.
```


Licensed Material-Property of Data General Corporation

In Example 4-9, our input file is a data file opened on channel C%, its byte offset is 0, its maximum number of records is 100, and its record size in bytes is 100. Our new index file is opened on channel CI%, its byte offset is 0, its maximum number of records (sectors) is 8, its record size in bytes is always 512, its blocking factor is 36 keys per block, it allows duplicate keys, it checks for deleted records in the data file, its number of fields is 2, and the first key starts at 1 and ends at 4, and the next one starts at 5 and ends at 10. IBUILD returns an error code using an STMA 2,1, ERCODE to pass the error to your program for you to check. For more information about IBUILD, see Chapter 7.

TBUILD

TBUILD is a utility program that will build a temporary index (tag file) from either a data file (with key or keys imbedded in the data record) or an index file.

Tag file records hold only one key and record pointer, but each key entry can have multiple key fields concatenated to form one key. See Figure 4-11 to see what a tag file key entry looks like:

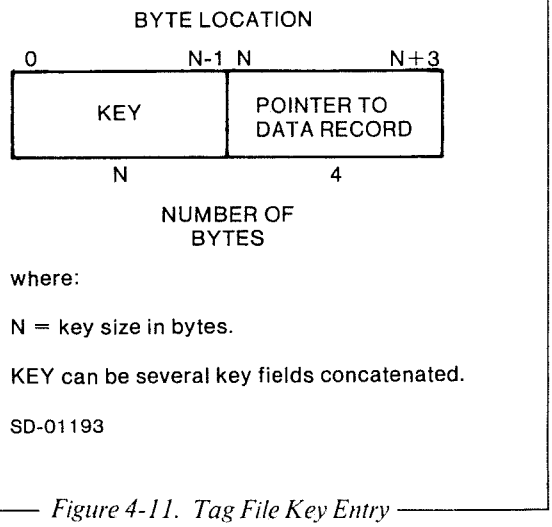


Figure 4-11. Tag File Key Entry

Tag files can be sorted fast (using QFILESORT), eliminating the need to sort data files. You can create a tag file for a data file and sort it faster than you can sort a data file. INDEXBLD calls TBUILD to create a tag file for a data file, then it builds the index file from the tag file.

Make your input file to TBUILD a data file with the keys imbedded in the data record, and be sure the data file is dynamically allocated. The record pointer is computed based on the record number accessed (relative to 0) in the data file, and TBUILD will optionally check for deleted records in the data file. You can also make the input file another index file. You must have already created and opened the input file and the new tag file for use.

Use TBUILD by building a string of input arguments and sending it into common area with a BLOCK WRITE statement. The input arguments for the string are really the same as the input arguments for IBUILD. The differences are: for the "number of key fields" argument, a value of -1 indicates an index file as the input file, instead of the -2 value used in the IBUILD input arguments (because the number of key fields is irrelevant if the input file is an index file); the channel number of the new index file for IBUILD is the channel number of the new tag file for TBUILD; and, TBUILD returns the string in common changing only the 4 bytes from locations 25 to 28 to the actual number of records in the tag file. See the description of TBUILD in Chapter 7 for more information and examples.

XBUILD

XBUILD builds an index from a data or tag file, using KADDs. This method is usually faster than IBUILD when the data file is not sorted. Blocking factors are ignored if you specify them and a blocking factor of approximately 75% is actually used.

The file you input to XBUILD, if a data file, must be dynamically allocated, and its records must contain the key. You must have already created and opened your input file and the new index for use.

The record pointer for the key is computed based on the record number (relative to 0) accessed in the data file, and XBUILD will optionally skip deleted records in the data file.

Use XBUILD by building a string of input arguments and passing the string to the common area with a BLOCK WRITE statement. The input arguments are the same as for IBUILD (see IBUILD, preceding this). For more information and examples, see the description of XBUILD in Chapter 7.

The next section tells you how to access the files you set up.

Licensed Material-Property of Data General Corporation

File Access

File access includes inputting from a file, outputting to a file, positioning to a byte or record within a file, deleting records in a file, finding available records in a file, finding a key, adding a key, and deleting a key from an index file, and locking files.

File Maintenance also accesses files, using keyboard operations, and it is discussed separately in Chapter 5. Screen Maintenance accesses a special type of file called a screen file, and it is discussed in detail in Chapter 6. EDIT is a utility program that inputs and outputs ASCII text files -- you can find EDIT in Chapter 7. INFOS files and their access methods are completely different from RDOS/DOS type files and regular Business BASIC file access, so they are handled separately in Chapter 9.

Input/Output Variables

Whenever you input to a file or output from a file, the amount of each data transfer depends on the size of the variable you use for the data. Regular numeric variables, i.e., those without the % sign, or the # sign, or the $ sign, transfer 4 bytes (32 bits) of information whenever you input or output. Simple numeric variables can hold values ranging from -2,147,483,647 to +2,147,483,647. Single precision variables ending with a % sign only transfer 2 bytes (16 bits) when you input from a file or output to a file, even though they are internally stored as 4 bytes, with the left 2 bytes unused. Single precision variables can hold values ranging from -32767 to +32767. Triple precision variables ending with a # sign are optional, and you can only use them if they were specified in the in the BSG dialogue during system generation. They allow 6-byte transfers when you input or output.

String variables ending with a $ sign transfer their entire DIMensioned length, regardless of their current length. If you write a string variable to a file, the file will contain the entire DIMensioned string, padded with nulls. If you read data from a file into a string variable, the DIMensioned length of the string variable will determine the number of bytes transferred to fill the string variable. For example, it is normally good practice to DIMension a string variable to equal the total number of bytes in a data record, so you can easily read the entire record into one string variable. Substrings are an exception: a substring only transfers its actual length (given in subscripts) on a write and only receives its actual length on a read.

File Access Keywords

Although we describe all BASIC statements in Chapter 3, we touch on them briefly in this section to show various techniques at work. Here's an entire list of them:

BLOCK READ	Input an entire block of binary data (512 bytes) from any type of file. Block I/O is fast but dangerous to the unaware. Never mix block I/O with regular I/O to the same file.
BLOCK WRITE	Output entire block of binary data (512 bytes) to any type of file. Use block I/O to transfer files or do special fast dumps, etc.
INPUT FILE	Input characters from a character format file.
KADD	Add a key to an index file.
KDEL	Delete a key from an index file.
KFIND	Find any key in an index. Will find an equal or greater value.
KNEXT	Find the next key after a KFIND. Use this to read an index file sequentially from any point.
MTDIO	Direct mag tape input/output.
POSITION FILE	Position to any byte in any type of file except sequential files.
PRINT FILE [USING]	Output characters to a character format file, optionally using a format.
READ FILE	Input binary data from any type of file.
TINPUT FILE	Timed INPUT FILE, allowing a specific amount of time for input.
WRITE FILE	Output binary data to any type of file.

Licensed Material-Property of Data General Corporation

Other file access statements that you can also use as commands are described briefly here in detail in Chapter 3. They are:

CLOSE [FILE]	Close all open files or a specific file.
DELETE	Delete a file from disk.
LOCK/UNLOCK	Lock and unlock records or blocks in a file.
OPEN FILE	Open a file for access and assign to it a channel number for access routines to use.
RENAME	Rename a directory entry filename.

Some BASIC CLI commands are also used with files, and they are described in detail in Chapter 8:

APPEND	Append a file to another file.
CCONT	Create a contiguous file with a specific number of contiguous blocks.
CRAND	Create a random file.
DELETE	Delete a file, directory, or partition.
FILCOM	Compare two files.
FILES	Display filenames in current directory.
LIBRARY	Display filenames in $LIB directory.
LINK	Link an alternate name (alias) to a file.
LSTCOM	Compare two ASCII files.
LSTMERGE	Merge two ASCII files.
MOVE	Move files from one directory to another.
PRINT	Print a text file, or send to output device.
TYPE	Display a text file at your terminal.

Subroutines used with files and file access are described in detail in Chapter 7, and touched on briefly in this section. They are:

DELREC.SL	Delete a record in a dynamically allocated file. Updates record 0 of file.
FINDFILE.SL	Builds the file characteristics array C1 for use in program with dynamic routines.
GETREC.SL	Get an available record. Uses and updates record 0 of a dynamically allocated file.
INITINDEX.SL	Do an INITFILE for an index. Buiilds block 0 and block 1 and makes a PARAM entry if needed.
POSFL.SL	Position to a record or a byte within a record of a dynamically allocated file. Uses the C1 array.

Utility programs used with files and file access are also described in detail in Chapter 7, and touched on briefly in this section. They are:

EDIT	Create, edit, input, and output ASCII text files.
FILESORT	Sort a data file.
IBUILD	Build an index file from a tag file or sorted data file.
INDEXBLD	Build an index file from a data file, tag file or another index file.
INITFILE	Create and build record 0 of a dynamically allocated file or create and build block 0 of an index file, making a PARAM entry if needed.
LOCKS	Display user's locks.
QFILESORT	Quick file sort in ascending order.
RELINK	Clean up, or reestablish, a dynamically allocated file's linked-available-records.
TBUILD	Build a tag file (temporary sorted index).
UCHANS	Display the user's file channel assignments.
SCHANS	Display the system channel assignments.
XBUILD	Build an index from a data file using KADDs.

Licensed Material-Property of Data General Corporation

You can also use three numeric functions in conjunction with file I/O, and these are described in Chapter 3:

EOF End of file. Returns a value of 1 if file is at end of file, if not it returns a 0.

GPOS The current position (in bytes) of the pointer to a file. The pointer is moved by a POSITION FILE statement or a POSFL.SL subroutine execution. Returns a -1 if file is not opened.

SYS Returns system information.

Here are the types of access allowed to data and index files: Sequential, (direct (or random), linked-available-record, block and ISAM (index files only). We'll discuss each separately in this section, and also discuss file locking.

Sequential Access

Sequential access is simply inputting and outputting consecutive records. You cannot position to the tenth record of a sequential file without inputting or outputting the first nine records. Sequential access is not meant to locate records in a file; it is meant to be fast and reliable for inputting and outputting entire files, inputting and outputting ASCII text (or any type of text) files, doing fast block I/O, and fast mag tape I/O.

Whenever you use INPUT FILE [USING] or PRINT FILE [USING], you perform a sequential access. Whenever you try to access a sequentially-organized file using READs, WRITEs, BLOCK READs, or BLOCK WRITEs, you perform a sequential access. You can't position yourself to a byte or record in a sequential file -- you can only start at the beginning and input or output data up to the byte or record you want. You can, however, sequentially access random and contiguous files by positioning to the first byte and inputting or outputting repeatedly. Any file that you can access randomly (directly) or access using ISAM, you can also access sequentially. Text files that use PRINT FILE and INPUT FILE statements can only be sequential, and all text input/output devices (line printer, paper tape, etc.) except tape drives (a special case, handled by MTDIO) are sequential. Also, any files you create by using the EDIT utility are sequential text files.

Example 4-10 shows a sequential access of a data file. This partial program starts at the beginning of the file and reads records until it finds a value of 0 in the first two bytes of a record:

Example 4-10

```
.
.
.
0040 DIM REC$(126)                  :This file has 128-byte records, but the
.                                   :first 2 bytes will be captured by STATUS%,
.                                   :and the remaining 126 bytes by REC$.
.
0070 OPEN FILE(2,0),"CUST"
0080 POSITION FILE(2,0)             :Position to byte 0 of file on channel 2.
0090 READ FILE(2),STATUS%,REC$      :STATUS% will capture first 2 bytes.
0100 IF STATUS%=0 THEN GOTO 0200          :Check for a 0 value.
0110 IF EOF(2)=1 THEN GOTO 0300           :If at end of file, go to 300.
0120 GOTO 0090                            :Read next record.
0200 PRINT "STATUS IS ZERO."
0210 LET P=GPOS(2)                        :Get current file position.
0220 LET P=P-128                          :Go back 128 bytes.
0230 POSITION FILE (2,P)                  :Position to beginning of record
.                                         :to process record.
.
.
```


Licensed Material-Property of Data General Corporation

You can perform sequential block access using BLOCK READs and/or BLOCK WRITEs (see Block I/O), and you can sequentially access an index file by KFINDing a null key and using consecutive KFINDs or KADDs (see ISAM).

Direct (Random) Access

Direct access involves using the POSITION FILE statement and the GPOS function. You can position yourself to any byte in a random or contiguous file with a POSITION FILE statement. You can find out where you are using the GPOS function, which will return the current byte position in the file. This is a simple and direct access for random and contiguous files, but if your files are dynamically allocated with a record 0 for header information, you can use linked-available-record access, described next.

You can position to a byte, or to a record by multiplying the number of bytes per record by the number of the record you want. You should only use READ FILE or WRITE FILE to input from or output to a file using direct access. BLOCK READ FILE and BLOCK WRITE FILE are discussed separately in this section, and MTDIO (mag tape I/O) is a special case of direct access also discussed separately. All of these statements can be found with more detail and examples in Chapter 3.

When you read or write a record to a file, the size of the data transferred depends on the size of the variable for the data. If, for example, you have a data record of 40 bytes, where the first 2 bytes are a signed numeric status flag, the next 4 bytes are a signed number, and the remaining 32 bytes are string values, you could read record 10 using POSITION FILE to get there and READ FILE to read the data into variables:

```
0050 DIM STR$(32)
0060 POSITION FILE (2,10*40)
```

(10*40 is the number of bytes from byte 0 to the beginning of record 10, since there are 40 bytes per record.)

```
0070 READ FILE (2),STATUS%,NUM,STR$
```

(This statement reads 2 bytes into STATUS%, 4 bytes into NUM, and the DIMensioned length of STR$ (32 bytes) into STR$.)

This method is fine for retrieving signed numbers and separate data items, but what if an interrupt occurred while the data transfers were happening? You'd lose some data, because even if you CONtinue execution, you cannot re-execute *part* of a statement, only the whole statement. If the interrupt occurred after the STATUS% transfer but before the NUM transfer, you wouldn't want to execute the READ FILE again because you'd be positioned to the wrong byte. You'd have to re-position and then reread.

There are several ways to avoid this problem. You can disable interrupts with an STMA, or handle them with an ON IKEY. If that's too hard, consider this: if only one transfer occurred in a READ FILE (or WRITE FILE), then it may not have interrupted -- it either happens or it doesn't, and that is easy enough for you to check. That's why we usually DIMension a string to the size of the record, and read or write the entire string.

One problem when DIMensioning strings is that you cannot extract signed numerics from a string accurately without coding an explicit routine to handle them. We solve that problem by providing the routine below which uses logical ANDs and ORs, and there are formulas for routinely-used 2-byte transfers and abnormal 1- and 3-byte transfers. For 4- and 6-byte transfers (double and triple precision integers), you don't have this problem -- you can extract them accurately from a string. Example 4-11 shows a read of a record using one large string for all the data and using a routine to extract STATUS% from the string and make it accurately signed.

Example 4-11

```
0050 DIM REC$(40)
0060 POSITION FILE (2,10*40)
0070 READ FILE (2), REC$

(The entire record is in REC$.)

0080 LET STATUS%=ASC(REC$(1,2))

(STATUS% is assigned the numeric value of bytes 1-2 in REC$.)

0090 LET STATUS%=OR(STATUS%,-AND(STATUS%,32768))

(This formula will give you an accurate sign for the value of STATUS%.)
```


Licensed Material-Property of Data General Corporation

You should define your formula in a user DEFined function, as we have done below. AND and OR are described in Chapter 3. The constant values in the formulas equal 1+ the highest value allowed in the number of bytes (e.g., 32768 equals 1+32768, the highest number allowed in 2 bytes). Here are the formulas, where VALUE is the variable whose sign needs to be corrected:

For 1 byte transfers:

```
DEF FNA(VALUE)=OR(VALUE,-AND(VALUE,128))
```

For 2 byte transfers:

```
DEF FNB(VALUE)=OR(VALUE,-AND(VALUE,32768))
```

For 3 byte transfers:

```
DEF FNC(VALUE)=OR(VALUE,-AND(VALUE,8388608))
```

Linked-Available-Record Access

Dynamically allocated files use linked-available-record access. Linked-available-records are records that were deleted and have a status (first 2 bytes of record) of 0. They form a chain of records that are available for you to WRITE to. Record 0 of a dynamically allocated file contains the information needed to use linked-available-record access. You don't have to access record 0; the subroutines provided with Business BASIC for linked-available-record access do it for you.

The subroutines supplied are described in Chapter 7, and they are: GETREC.SL, DELREC.SL, and POSFL.SL. DELREC.SL deletes a record for you and puts it on the linked-available-record chain. GETREC.SL gets an available record for you to write to. POSFL.SL positions you to any record or any byte within a record. You need the file characteristics array C1 to use these subroutines.

When using subfiles, it is usually easier to make them dynamically allocated and use linked-available-record access (with a C1 array) than to try to access the subfiles directly.

To Position to a Record

When you OPEN a file using the OPEN utility, you put the channel number in C1 array. The C1 array's row dimension corresponds to the logical file number. When you OPEN a file using the OPEN FILE statement, you assign a channel number to the file, and you should put that channel number in the C1 array. POSFL.SL looks for the channel number in C1 associated with the logical file number you specify to POSFL.SL. POSFL.SL expects to see two variables with assigned values in your program: F% for the logical file number, and R1 for the record number you want. Optionally, you can specify a byte location in the record R1 with the variable V%.

There are two entry points to POSFL.SL: you can GOSUB to 9610 to position to a record R1: if you specify V%, you can GOSUB to 9612 to position to a byte within record R1, where V%=0 refers to the first byte of the record. POSFL.SL returns after executing with three values: C%, for the channel number of the file to be used with READ FILE or WRITE FILE: R9, for the byte position in the physical file at the start of record R1: and R8, for the byte position in the logical file (different from R9 if logical file is a subfile) of the start of record R1. You can use R9 or R8 with a POSITION FILE statement to do a fast rewrite of the record.

Example 4-12 shows one use for POSFL.SL. We position to a record, read the record, and use POSITION FILE with R9 to go back to rewrite the record. Note that we left in the code to open the files and fill the C1 array.

Licensed Material-Property of Data General Corporation

Example 4-12

```
0010 DIM X$(512),C1(2,3),REC$(48)        :Record size of SUB2 is 48.
0020 LET X$="SUB1,5,SUB2,5,PHYS,6",FILL$(0)
0030 BLOCK WRITE X$                      :Send logical file info into common.
0040 SWAP "OPEN"                         :OPEN will return X$ with C1 array.
0050 BLOCK READ X$                       :Retrieve info from common area.
0060 LET K=1                             :Pointer to first element in string.
0070 FOR I=0 TO 2                        :For each logical file, 0, 1 and 2,
0080   FOR J=0 TO 3                      :and for each dimension of C1 array,
0090     LET C1(I,J)=ASC(X$(K,K+3))      :extract element, put in C1 array.
0100     LET K=K+4                       :Bump pointer 4 bytes.
0110   NEXT J
0120 NEXT I
0130 INPUT "RECORD NUMBER OF SUB2 TO BE REWRITTEN: ",NUM
0140 LET R1=NUM                          :Give POSFL.SL a record number R1.
0150 LET F%=1                            :F% used by POSFL.SL for logical file.
0160 GOSUB 9610                          :Position to record R1 in file F%
                                         :using POSFL.SL, returns C%, R8 and R9.
0170 READ FILE (C%),REC$                 :C% is channel number.
0180 DIM NEWREC$(48)                     :For new record.
.                                :Code to build NEWREC$ for new record g.
.
.
0210 POSITION FILE (C%,R9)       :Use R9 returned by POSFL.SL to position to
0220 WRITE FILE (C%),NEWREC$     :and rewrite record R1.
.
.
.
```

Example 4-13

```
.
.
.
0130 INPUT "RECORD NUMBER OF SUB2 TO BE REWRITTEN: ",NUM
0135 INPUT "BYTE OFFSET, OR 0 IF BEGINNING OF RECORD: ",BYT
0140 LET R1=NUM
0145 LET V%=BYT
0150 LET F%=1
0160 GOSUB 9612                  :Alternate entry to POSFL.SL if you use V%.
0170 READ FILE (C%),REC$(BYT+1,48)       :Read using byte offset into string.
.
.
```

Example 4-13 shows one use of POSFL.SL to position to a byte within record R1. V% is the byte position relative to 0 (i.e., V%=0 is the first byte of the record).

In Example 4-13, you'd only want to read the part of the record specified from byte position BYT to the end. Therefore, you read the data into a substring instead of the whole string, and you set the substring's size to start at the byte location BYT plus 1 because records begin at byte 0 and strings begin at byte 1.

Licensed Material-Property of Data General Corporation

To Delete a Record

Use DELREC.SL. DELREC.SL automatically updates record 0 and deletes the record by setting its status (first 2 bytes) equal to 0. The deleted record then joins the chain of deleted and available records so eventually you can reuse that space. DELREC.SL requires the variables, R1 and F%, be assigned as with POSFL.SL, and it requires the file characteristics (C1) array. You assign a logical file number to F%, a record number to R1, and you supply the C1 array. You enter DELREC.SL at entry point 8600. DELREC.SL doesn't return any values, but it uses scratch variables X0%, Y0%, X0, Y0, and Z0 -- you can't use these variable names for other variables in your program. You don't have to worry about initializing or dealing at all with the scratch variables. When DELREC.SL executes, it automatically calls POSFL.SL to position to the desired record. Here is a sample usage of DELREC.SL:

```
.
.
.
0130 INPUT "REC TO BE DELETED: ",NUM
0140 LET R1=NUM
0150 LET F%=1
0160 GOSUB 8600
.
.
.
```

SUB2 is logical file number 1 in the C1 array. Line 160 GOSUBs to DELREC.SL to delete the record.

To Write a Record

Use GETREC.SL to get an available record to write to. GETREC.SL finds an available record from the deleted record chain, updates record 0, and returns to you the record number in R1 that you use with POSFL.SL to position to the record before writing to it.

GETREC.SL allocates a new record in random files if there are no more records in the deleted record chain. If you run out of deleted records in a contiguous file and you've used up all the space allotted to the contiguous file, you have to copy your contiguous file into a larger contiguous file or to a random file.

GETREC.SL needs a variable, F%, for the logical file number and it expects a file characteristics (C1) array. It uses the same scratch variables DELREC.SL uses, and you can't use those variable names for other variables in your program. GETREC.SL returns the variable, R1, with the record number you use with POSFL.SL. After executing GETREC.SL (8400 is the only entry point), you execute POSFL.SL to position to the record using R1 returned by GETREC.SL, and then you WRITE FILE the new record.

Example 4-14 shows a partial update session where a record is deleted using DELREC.SL and a new one is added using GETREC.SL. This is a good update technique if placement of new records doesn't matter -- and it wouldn't matter if you have an index to the data file, as long as you also KADD the new key to the index (see "ISAM" later in this section).

Example 4-14

```
.
.
.
0130 INPUT "RECORD OF SUB2 TO BE DELETED: ",NUM
0140 LET R1=NUM
0150 LET F%=1                    :SUB2 is logical file 1 in C1 array.
0160 GOSUB 8600                  :Go to DELREC.SL to delete record.
0170 GOSUB 8400                  :Go to GETREC.SL to find next available record.
0180 GOSUB 9610                  :Go to POSFL.SL using R1 returned by GETREC.SL.
0190 WRITE FILE (C%),NEWREC$     :C% is channel number returned by POSFL.SL.
.
.
.
```


Licensed Material-Property of Data General Corporation

Block Input/Output

Use BLOCK READ FILE and BLOCK WRITE FILE, for block I/O to transfer 512-byte blocks (sectors). The size of the variable you use with a block statement determines the number of blocks transferred. Your variable must be able to hold at least 512 bytes -- that's why you can only use string variables and arrays with BLOCK READ and BLOCK WRITE.

Block I/O is fast and reliable, but it is not buffered; since all other I/O is buffered, you can't mix block I/O with regular I/O to the same file at the same time. Block I/O can be a random (direct) access because in a BLOCK READ FILE or BLOCK WRITE FILE statement you can specify the block number in the file where you want to start the I/O operation. You have to read/write sequentially from that point, however, so you can also perform a sequential access from the first block in the file. GPOS does not work when you use block I/O.

BLOCK READ FILE reads into a string variable or array as many full blocks as the string variable or array can hold. If string variable A$ was DIMensioned to 1536 bytes, 3 blocks would be transferred: 1535-byte string variables would only transfer 2 blocks. Array elements can hold 4 bytes each, so array A needs to be DIMensioned to at least 128 items for 1-block transfers, 255 for 2 blocks, etc. Two-dimensional arrays are allowed; i.e., array A with 128 elements could be DIMensioned as A(3,31) with 4 rows and 32 columns (arrays begin at element 0 -- row 0 and column 0).

Conversely, if the string variable or array is larger than the amount of data in the block, the string variable or array will be filled with nulls. These rules also apply when you write strings or arrays to a file with a BLOCK WRITE FILE statement.

BLOCK READ and BLOCK WRITE (i.e., without the keyword FILE) are special cases of block I/O that write to or read from a common area set up for each terminal. The common area for each job is one block (512 bytes) long and is always open and ready for use. You can put data into the common area with a BLOCK WRITE or BLOCK WRITE FILE (16) statement, since channel number 16 is a dummy channel for your terminal. You can then SWAP, CHAIN, or RUN a new program, and your data is still there in the common area for you to retrieve with a BLOCK READ or BLOCK READ FILE (16) statement. This is the easiest way to pass information to other programs, but you can only pass 512 bytes of information at a time, in the form of a string or array. You should always fill the string or array with nulls when you write them (if smaller than 512, fill to 512). You'll always need a 512-byte string or array to read the common area, and if the data is smaller than 512, BLOCK READ will pad the string or array with nulls.

Here are two examples: Example 4-15 uses the common area to pass information and retrieve file information from the OPEN utility program, and Example 4-16 copies the file "IN" to the file "OUT" by transferring 16 blocks at a time. More information about BLOCK READ and BLOCK WRITE can be found in Chapter 3.

Example 4-15

```
0010 DIM X$(512)                    :For one block.
0020 LET X$="SUB1,5,SUB2,5,PHYS,6",FILL$(0)
0030 BLOCK WRITE X$                 :Zip-- into common area,.
0040 SWAP "OPEN"
0050 BLOCK READ X$                  :We read what OPEN sent to common area.
.
.
.
```


Licensed Material-Property of Data General Corporation

Example 4-16

```
0010 DIM A$(8192)                    :For 16 blocks.
0020 OPEN FILE(0,4),"IN"             :Open files.
0030 OPEN FILE(1,0),"OUT"
0040 LET B=0                         :Initialize block number.
0050 BLOCK READ FILE(0,B),A$         :Read up to 16 blocks starting at B.
0060 BLOCK WRITE FILE(1,B),A$(1,LEN(A$))        :Write blocks just read,
                                     :but only write current length of A$ in
                                     :case we have less than 16 blocks of data.
0070 IF EOF(0) THEN GOTO 0100        :Test for an end of file.
0080 LET B=B+16                      :Bump starting block number.
0090 GOTO 0050                       :Go back and read 16 more blocks.
0100 CLOSE                           :If at end of file, close all files.
0110 STOP
```

ISAM (Index File Access)

ISAM means Indexed Sequential Access Method, and it allows you to find any key in an index, add any key, delete any key, and read an index sequentially from a found key. Each key points to a data record in a data file, and you can have many index files for one data file, many keys for one data record, and duplicate keys that point to different records.

Index files have to be created in special ways (see "Creating Index Files" in this chapter). They have a tree structure that reduces the amount of disk accesses needed to retrieve information, and they are maintained in a sorted order. The ISAM statements are: KADD (add a key), KFIND (find a key), KNEXT (find the next key), and KDEL (delete a key). These statements use block 0 of an index file, so you have to follow the conventions when creating your index file.

Each statement needs three strings and a record pointer variable. The first string is an index file descriptor string. It can be any size greater than 17 bytes, but it has to look like the diagram in Figure 4-12.

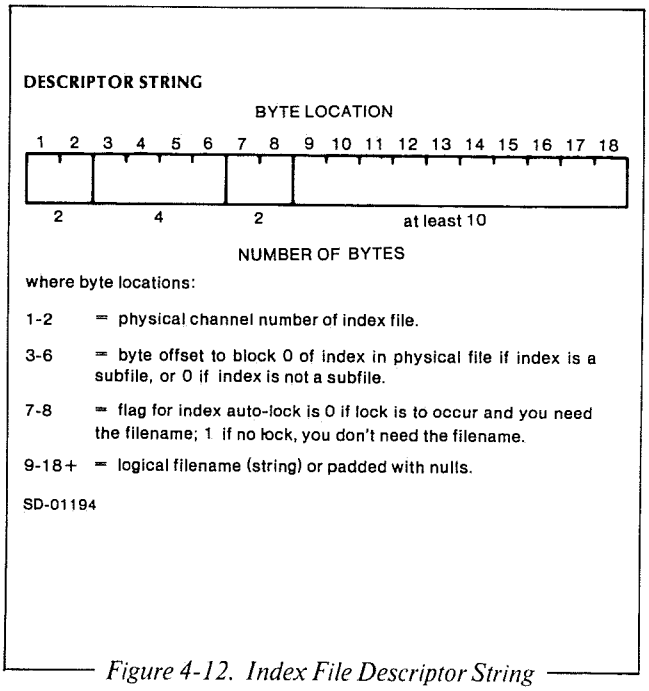

DESCRIPTOR STRING
BYTE LOCATION
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
2
4
2
at least 10
NUMBER OF BYTES
where byte locations:
1-2 = physical channel number of index file.
3-6 = byte offset to block 0 of index in physical file if index is a subfile, or 0 if index is not a subfile.
7-8 = flag for index auto-lock is 0 if lock is to occur and you need the filename; 1 if no lock, you don't need the filename.
9-18+ = logical filename (string) or padded with nulls.
SD-01194

Figure 4-12. Index File Descriptor String

Licensed Material-Property of Data General Corporation

You should set this string up early in your program. Most of the information can come directly from the file characteristics array (C1) if you're using it. Here's an example of filling descriptor string DESC$ with information from C1 array, with F% as the logical file number and F$ as the logical filename:

```
LET DESC$=CHR$(C1(F%,0),2),CHR$(C1(F%,1),4),
  CHR$(0,2),F$,FILL$(0)
```

The first 2 bytes of DESC$ contain the channel number of the physical file that holds the index file, taken from row F% and column 0 of the C1 array. The next 4 bytes contain the byte offset to block 0 of the index file within the physical file, also taken from the C1 array at row F% and column 1. The next 2 bytes now contain a flag of 0 to allow automatic locking of the index and the logical filename of the index file (needed to lock the index file) is in F$.

If your index file is not catalogued in the C1 array, then it is probably not a subfile, and you can fill your descriptor string (D$) like this (C% is channel number, INDEX is the logical filename):

```
LET D$=CHR$(C%,2),CHR$(0,4),CHR$(0,2),
  "INDEX",FILL$(0)
```

If you don't want automatic locking of the index, you don't need the logical filename, and you could have done this:

```
LET D$=CHR$(C%,2),CHR$(0,4),CHR$(1,2),FILL$(0)
```

Be sure to create one descriptor string for each index file to be accessed, because all of the ISAM statements use it.

The second string you need with each ISAM statement is a buffer string DIMensioned to at least 544 bytes. This string is used to hold an index block whenever you access an index file. You don't do anything to it but DIMension it.

The third string is the key itself. KADD and KDEL require you to supply the actual key value in the key string. KFIND and KNEXT require you to DIMension a key string to the maximum number of bytes per key, and those statements return a key value to you in the key string. Key entries to the same index file always have the same maximum number of bytes. Keys can be strings, numerics, concatenated strings, crammed strings (see the CRM$ function in Chapter 3), or any combination of these.

The record pointer variable is a 4-byte numeric variable holding the number of the data record that the key points to. You have to supply the record number for KADDs, but they are optional for KDELs if you have no duplicate keys in your index file. KFIND and KNEXT require a variable in the statement, but they return a record number for the variable. If KFIND or KNEXT returns to 0 for record number, you bumped into the end of the index. If KFIND of KNEXT returns a positive record number, you got an exact match for your key. If they return a negative record number, you got an approximate match that is slightly greater than the desired key.

To Add a Key

KADD adds the key string and record pointer to the index file described in the descriptor string. KADD searches the index to find the proper location for the key. If you didn't allow duplicate keys in the file KADD finds a duplicate, it sends the record pointer variable back to you with a value less than or equal to 0, and it doesn't add the new key.

In multiple-user systems you should allow automatic locking of the index file (set the flag to 0 in the descriptor string and supply a logical filename in the descriptor string), and you should lock the data record the new key points to (especially if you are adding a new data record). See also "File Locking" in this chapter.

Example 4-17 shows a partial program that writes a new data record and adds a new key. Note that we allow automatic locking of the index file by setting the flag in DESC$ to 0 and supplying the logical filename. Both the index file and the data file are catalogued in the C1 array, and we use GETREC.SL and POSFL.SL to write the new data record.

Licensed Material-Property of Data General Corporation

Example 4-17

```
0010 DIM DESC$(18),BUF$(544)     :Strings used with KADD.
.                                :Code for C1 array and OPEN utility goes here.
.                                :MASTER and INDEX files are in C1 array.
.                                :Fill DESC$ descriptor string here.
0200 DIM KEY$(10),REC$(48)       :Key length 10 bytes, record length 48 bytes.
0210 INPUT "TYPE KEY: ",KEY$
0220 INPUT "TYPE DATA: ",REC$    :Input record data and key.
.                                :Code to validate data and key goes here.
.
.
0300 LET F%=2                    :Logical file 2 is data file.
0310 LOCK 2,"MASTER",0,C1(F%,3)  :Lock record 0 of data file MASTER.
0320 GOSUB 8400                  :Go to GETREC.SL to get available record,
                                 :record number returned in R1.
0330 UNLOCK 2                    :Unlock lock #2,.
0340 GOSUB 9610                  :Position to record R1 using POSFL.SL.
0350 WRITE FILE(C%),REC$         :Write new record using C% returned by POSFL.SL;L.
0360 KADD DESC$,BUF$,KEY$,R1     :Add new key.
0370 IF R1<=0 THEN GOTO 0500     :If duplicates are not allowed and KADD found
.                                :one, or if an error occurred, KADD returns
.                                :R1 with a value of 0 or less.
.
```

To Delete a Key

KDEL deletes a key and pointer from the index file described in the descriptor string. You have to supply the key in a string, and if duplicate keys are allowed in the index file, you also have to supply the record number in a variable; if duplicate keys are not allowed, you supply a variable name only. Your record number variable is returned by KDEL with a value of 0 if no match of the key was found in the index, or with the value of the record pointer for the key if the delete was successful.

In a multiple-user system, you should allow automatic locking of the index file (set the flag in the descriptor string) and you should lock the data file if you are also deleting records. If you delete the key in the index first and then delete the record, nobody can access the record through the key before you delete the record.

The partial program in Example 4-18 uses automatic locking for the index file and LOCK/UNLOCK (described in "File Locking" in this chapter) for the data file. First KDEL deletes the key, then DELREC.SL deletes the data record. The DESC$ descriptor string and BUF$ buffer string were shown in previous examples. Note that we use KFIND to find the record pointer R1 for use in the program.

Licensed Material-Property of Data General Corporation

Example 4-18

```
.              :Code to set up files, dimension strings, C1 array, etc.
.              :DESC$ descriptor string must be filled with information
.              :about the index file.
0500 INPUT "KEY FOR RECORD TO BE DELETED: ",KEY$
0510 KFIND DESC$,BUF$,KEY$,R1          :Find key and R1 record pointer.
0520 IF R1<=0 THEN GOTO 0700           :If record doesn't exist, go to 700.
0530 LET F%=3                          :Logical file number for data file.
0540 GOSUB 9610                        :POSFL.SL uses F% and R1.
0550 LOCK 3,"MASTER",R8,C1(F%,3)       :R8 returned by POSFL.SL.
0560 READ FILE(C%),REC$                :C%  returned by POSFL.SL.
.                                :Do record checking here.
.
.
0600 KDEL DESC$,BUF$,KEY$,R1     :Delete key entry from index file.
0610 LOCK 4,"MASTER",0,C1(F%,3)  :Lock record 0 of data file.
0620 GOSUB 8600                  :Delete record from data file.
0630 UNLOCK                      :Safe to unlock both deleted record and
.                                :record 0.
.
.
```

To Find a Key

KFIND searches the index file (described in the descriptor string) for a match to the key you supply. If KFIND finds an exact match, it returns the data record pointer associated with the key. If KFIND can't find the exact key, it finds a key with a value slightly greater than the key you specified, and returns the negative value of the record pointer associated with the found key. For example, if you specify key "ABC" and there is no "ABC" but there are two keys, "ABCA" and "ABCB", KFIND would find key "ABCA," and a subsequent KNEXT would find "ABCB". If KFIND can't find a key value equal to or greater than the key you supply, it will return a value of 0 for the record pointer.

KFIND was used in the previous example. Example 4-19 is another example of KFIND.

Licensed Material-Property of Data General Corporation

Example 4-19

```
.                          :Code for C1 array, OPEN utility, dimensioning
.                          :array and strings, setting up DESC$, etc. goes here.
.
0800 INPUT "NAME? ",KEY$
0810 KFIND DESC$,BUF$,KEY$,R1    :Search index described in DESC$ and return R1.
0820 IF R1<=0 THEN GOTO 0700     :If no key or only approximate, go to 700.
0830 LET F%=3                    :Logical file number for data file.
0840 GOSUB 9610                  :Position to data record R1.
.
.
```

To Find the Next Key

KNEXT finds the next key in sequence after a KFIND in the index file which is described in the descriptor string. KNEXT needs the buffer string (BUF$ in these examples), so don't change the buffer string between KFIND and KNEXT or between KNEXTs. KNEXT returns the record pointer associated with the key found. If KNEXT bumps into the end of the index, it returns a 0 for the record pointer.

KNEXT is used mostly to read an index sequentially from a given key. Use it also to find all occurrences of duplicate keys. To read an index sequentially, the proper technique is to use KFIND with a null key to return the first key in the index, and then use KNEXTs, as we do in Example 4-20.

Example 4-20

```
.                          :Code to set up C1 array, use OPEN utility, set up
.                          :DESC$ string, etc. goes here.
.
0100 LET KEY$=""                 :Null key to get first entry in index file.
0110 KFIND DESC$,BUF$,KEY$,R1    :Find first key entry.
0120 LET R1=ABS(R1)              :Negative R1 expected for approximate match.
0130 GOTO 0150                   :Process key and pointer for first record
                                 :before calling KNEXT.
0140 KNEXT DESC$,BUF$,KEY$,R1    :Get next key in secuence.
0150 IF R1=0 THEN GOTO 0600      :If at end of index, go to 600.
0160 LET F%=3                    :Logical file number for data file.
0170 GOSUB 9610                  :POSFL.SL uses F%, positions to data record R1.
0180 READ FILE(C%),REC$          :C% returned by POSFL.SL; read record.
.                                :Process record here.
.
.
0200 GOTO 0140                   :Go back to KNEXT to get next key.
.
.
.
```


Licensed Material-Property of Data General Corporation

When you access two index files, you need two separate buffer strings (BUF$ in this example) for each index file if you are using KNEXTs. KADD, KDEL, KFIND, and KNEXT are all described in Chapter 3.

File Locking

Business BASIC does not actually *lock* files, it sets up a lock table with the names of the files "locked" and the block numbers where the lock exists. It is essentially a communications device -- when you lock a file, it searches the lock table to see if some other user has already locked the file. For this reason all users of a system must agree on what filenames to use: logical or physical (we recommend logical, especially if you use subfiles -- then you only need to lock subfiles).

You lock and unlock files using the statements LOCK and UNLOCK, which are described in Chapter 3. You don't have to lock a file if you opened it for exclusive use (see OPEN FILE or the OPEN utility), you don't have to specifically lock index files -- they have automatic locking, and you only have to set the flag in the descriptor string (see "ISAM" in this chapter).

LOCK is not a record lock but a block lock. In the LOCK statement you specify a starting byte for the lock area and the size of the lock area (usually the beginning of a record and the size of the record). LOCK automatically adds the size to the starting location and comes up with an ending location, then divides the starting and ending locations by 512 to get the block (sector) numbers of the area to be locked. If the area you specified spans more than one 512-byte block, then any other block it touches will be included in the lock area.

LOCK then searches the lock table for the name you gave it, and if it doesn't find the name in the lock table, it puts it there for you. If the name is in the lock table, it checks to see if the lock areas are the same. If they are, your lock doesn't work, and your job suspends. If they aren't, your lock is successful. A successful lock is one that exists in the lock table. It is only removed from the lock table with an UNLOCK.

LOCKs only work if *everybody* uses them. If you LOCK a block in a file, there is no way to stop another user from accessing that block unless that user tried to do a LOCK. Also, if you don't follow a naming convention, you might try to LOCK "MASTER" which is a physical file and someone else might try to LOCK "SUB1" which exists in MASTER -- both LOCKs would work, but you'd still clobber each other if you accessed the same place. That's why we recommend that you use logical filenames -- if you LOCK "SUB1", no one else can LOCK "SUB1". Each system should have conventions to follow when locking. For example, everybody must try to LOCK records before accessing them, everybody should use logical filenames, etc.

When a LOCK fails, it suspends your job until the other user UNLOCKs the file area you want. If that other user is trying to LOCK something you already LOCKed, the other user's job also suspends, and since the other user can't issue an UNLOCK if he/she is suspended, a deadly embrace situation occurs. To avoid deadly embraces, use a variable with the LOCK statement to assign a time for the system to wait for an UNLOCK to occur. When a LOCK is unsuccessful and you specified a time (in tenths of seconds), it will suspend for that amount of time only. If an UNLOCK did not occur, it will return execution to the program, and the variable you assigned the time to would return with a value of 57 to indicate an unsuccessful lock.

LOCK statements also require an identification number that is local to the job performing the lock identifiers can be the same for different jobs (terminals). If you attempt to perform two LOCK statements at the same terminal with the same identifier, an error will occur, the lock will not be successful, and a 56 would be returned in the variable with LOCK. You can find out what locks have been performed at your terminal by using the LOCKS utility program which displays the current locks for your job (terminal). You can reuse an identifier once you UNLOCK it and free it for use. CLOSE, NEW, and BYE also perform automatic UNLOCKs.

The LOCK and UNLOCK statements are described in detail in Chapter 3. Refer to previous examples in this section for sample uses of LOCK and UNLOCK, or look at the examples in Chapter 3.

End of Chapter

Licensed Material-Property of Data General Corporation

Chapter 5
File Maintenance

The File Maintenance (FM) utility is made up of a group of utility programs written in Business BASIC that give you keyboard mode access to any file set up using File Maintenance. In keyboard mode you can open a file, find a data record using index file keys or using its record number, change a record, delete a record, and add a record. You can also print data records using the FM format and the utilities FMPRINT, and FMTABPRINT for table files, and FMLOG for a log of all changes to a file. Because these utilities use FM they are, therefore, described only in this chapter.

You can set up a direct (random) access file or linked-available-record file (dynamic allocation) in FM using an ISAM index file associated with it. You can have up to three index files pointing to one data file. You can locate a record either by its record number, or by up to three separate keys (from index files). You can print quick reports of all the activities of a file (log file). You can also print the entire data file in an FM format, or print selected portions of a data file based on the first and last keys you select. You can even create and fill a data file from scratch using FM to key in each record and add each new key.

Multiple users can use FM to access a single data file, and read and update records within that file because FM locks records in use to keep users from accessing the same record at the same time. User programs can also access the data file at the same time if they use the BASIC file locking facility (described in Chapter 4) and conform to FM conventions.

You can have multiple formats for one type of data record, and you can have many types of data records in the same file using FM. You can use SM screen files to display records (see Chapter 6), but FM has its own screen that you can define using FM. FM works for the 6053 terminals to produce high/low intensities and to allow use of function key control for faster access. There's another version of FM (roll-mode) for use with other terminals (like the 6012) that do not have function keys. We describe both access methods in this chapter. We'll show you step by step how to create and maintain a customer data file -- CUST -- and its associated table file, CUST.TB, its log file, CUST.LG, and its associated index files, CUSTI1, CUSTI2, and CUSTI3.

To be used with FM, a data file must have a table file created by FM. You must also create the index files and data file and catalog them in the PARAM file, which is described in Chapter 4. Note that you can use FM to access the PARAM file.

You can optionally set up data files that are protected from users by passwords. The user would have to type his/her account number and a valid identification sequence of up to nine digits to gain entry to the file using FM. You can, or course, set up a file with no password requirements that are open for use by anyone in the file's directory.

Setting up a Table File

To create a file using FM, you must CRAND or INITFILE the file, its index files, and log file (if you want a log file). You must also catalog the files in the PARAM file (INITFILE does this for you). Your data can be of any type, and your data file can be random or contiguous with direct (random) access and/or linked-available-record access (dynamic record allocation). FM supports both of these access methods for data and log files, and supports ISAM for index files. See Chapter 4 to find out more about creating such files.

NOTE: If you do not use linked-available-record access (dynamic allocation), you *still* have to have the first 2 bytes of each record reserved for record status (0 for deleted records, 1 or greater for active records). Your different record types in the file (if you want different record types) would also use this status flag.

A table file contains names and types of index and data files, heading information, record and file sizes, a valid users list, record format descriptions, field descriptions, and key descriptions.

A table file has 6 different record types that have predefined locations within the file. This is why you should use FM to create your table file. The record types are set up to point to other record types. Table 5-1 summarizes the record types in a table file.

Table 5-1. Record Types in a Table File

Record Type	Record Number(s) in Table File	Description
1 (control record)	0	Contains names of data and index files, number of record types, number of fields, number of pages, and pointers to other descriptions.
2 (control record cont.)	1	Heading, record, and file size of data file, and optional log file and screen filenames.
3 (user record)	2	Eight valid user IDs and passwords.
4 (format descriptor record)	3 - 12	Pointer to the first field descriptor record (type 5 record) with the screen format for each record type, and number of pages.
5 (field descriptor record)	Usually 16-400	Record for each field on each page for each type. You can have nine different types, 385 different fields, and up to seven pages per type.
6 (index descriptor record)	Usually 13-15	Record for each key to describe each key field.

Record types 1, 2, and 3 must be in records 0, 1, and 2, respectively. Record type 4 must start at record 3. Record types 5 and 6 can be anywhere else in the table file as long as the pointer to the record type points to the first record of that type, and subsequent records of that type follow the first record in sequential order.

Let's step through the process of creating a table file. We'll start by creating our data and index files using INITFILE (see Chapter 4 for the INITFILE dialogue). The data file -- CUST -- is in master file CUST at byte 0, has a record length of 128 bytes and a last record number 100. We have three index files: CUSTI1, CUSTI2, and CUSTI3 which reside in master file CUST, with 512-byte blocks, at byte position 0, 5120, and 10240, respectively, with a maximum of ten records (blocks) each. We also INITFILE a log file CUST.LG to have a record length of 146 bytes and a last record number 200. We use INITFILE because we need to have these files in the PARAM file. An alternate way would be to CRAND or CCONT the files and add them yourself using FM (see "Using FM" later in this chapter). Finally we create our table file -- CUST.TB -- using CRAND.

You start FM by typing RUN "FM", CHAIN "FM", SWAP "FM" (or just "FM"), or from the BASIC CLI by typing FM. The DATA FILE MAINTENANCE heading appears, with the question FILENAME: . To create a table file, you answer FM.TB. The File Maintenance parameters and record descriptions are in FM.TB; File Maintenance maintains itself using FM.TB. After you type FM.TB, FM then asks you for a *DATA FILE NAME:* You answer with the name of the table file you are creating for your data file *not* with the data file's name. Usually the table file is the data filename with a .TB extension -- our table file is CUST.TB:

```
*RUN "FM" )

DATA FILE MAINTENANCE REV 3.52 10/28/77

FILENAME: FM.TB )

DATA FILE NAME: CUST.TB )

FILE MAINTENANCE TABLE FILE

RECORD # ----------
```

If you're using roll-mode FM, you have to enter function commands to the FUNCTION query. If you're using regular FM on 6053 terminals, you can hit function keys for commands. Table 5-2 summarizes these commands, and Figure 5-1 shows a template you can use for your 6053 terminal function keys.

Licensed Material-Property of Data General Corporation

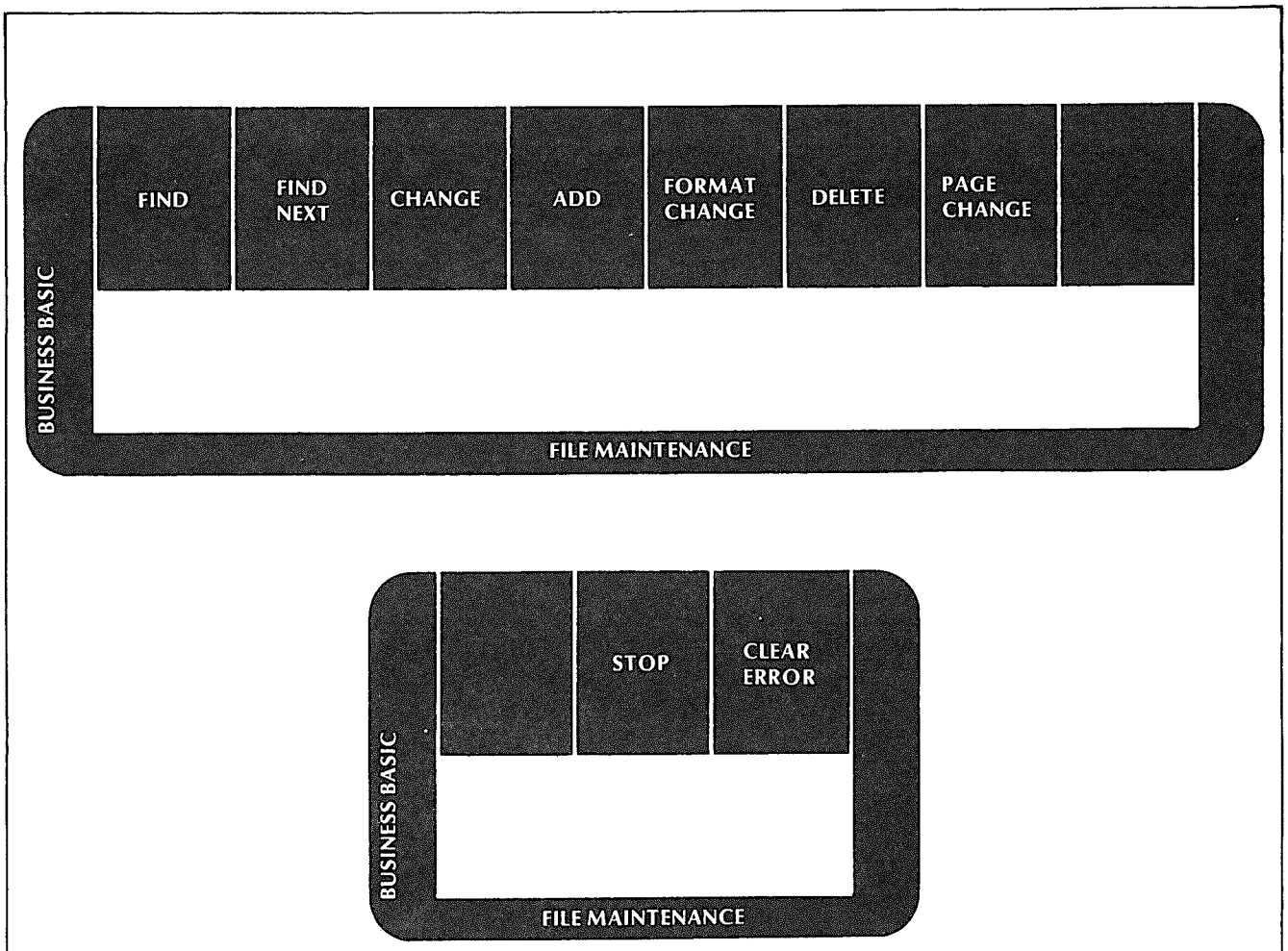


Figure 5-1. File Maintenance Templates

Licensed Material-Property of Data General Corporation

Table 5-2. FM Function Keys

Function	Key	Roll-Mode Command
Find a record	F1	FIND
Find next record	F2	NEXT
Change a record	F3	CHANGE
Add a new record	F4	ADD
Format change	F5	FORMAT
Delete a record	F6	DELETE
Page change	F7	PAGE
Stop FM	F10	STOP
Clear an error	F11	(not applicable)
Display record	not applicable	DISPLAY
Automatic display	not applicable	AUTO
Next field	Carriage return (CR)	Tab/CR key
Previous field	F12 (Backtab)	(not applicable)
Complete function	Home	(after last field)

You can follow our step by step method of creating table file records for CUST.TB using ADD and the FORMAT CHANGE keys. If you're in roll-mode, FUNCTION? should be displayed and you should answer FORMAT. If you're using regular FM, two square brackets ([]) appear at the top left of the screen, and a RECORD # field appears. Use the FORMAT CHANGE (F5) key, and the word FORM appears within the square brackets. The cursor would now position to the RECORD # field. It wants to know what type of record you're changing the format to. You should answer with a 1 for record type # 1. A type 1 record in a table file is the control record, and it's at record 0, but you have to input the *type* of record. You have to start building a table file from record type 1 and progress to record type 6.

Type 1 Records

When you type a 1, # 0 CONTROL RECORD appears with all the fields in the record. See Figure 5-2.

Licensed Material-Property of Data General Corporation

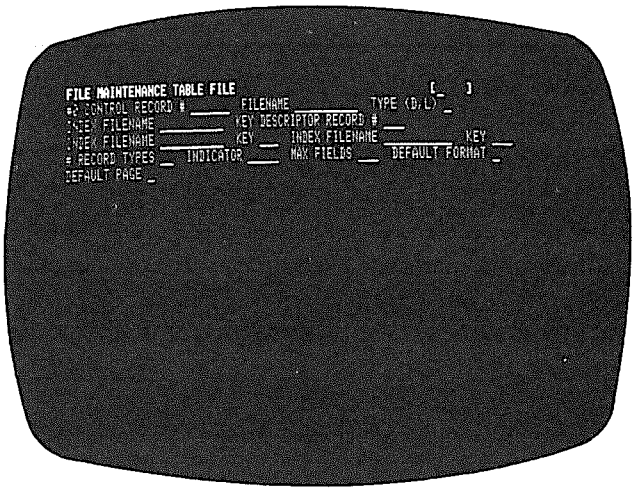


Figure 5-2. Type 1 Record

Now we enter control information for a type 1 record, as shown in Figure 5-3.

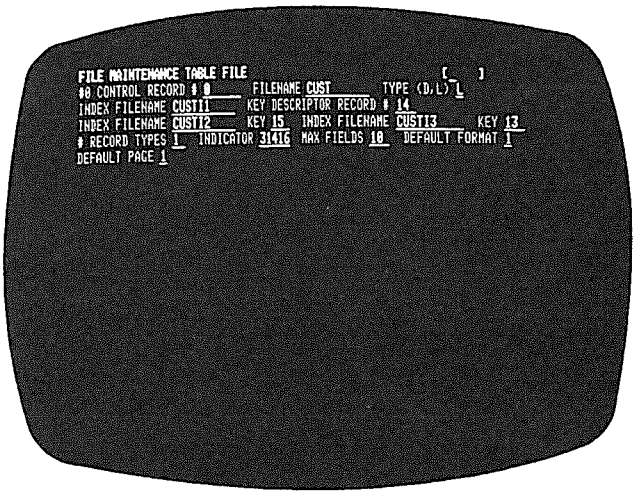


Figure 5-3. Type 1 Record with Control Information

Our answer to # 0 CONTROL RECORD # must be 0. The FILENAME indicates the data file, and we must enter the same name we use for logical filename in the PARAM file. We enter CUST.

TYPE:

We entered an L to signify that our data file is linked (linked-available-record format) for dynamic allocation. FM automatically allocates new space and reuses space freed by deleted records for this type of file, but the file must have already been INITFILEd (record 0 in the file must be set up). See Chapter 4 for a description of dynamic allocation. If you enter a D, you signify that your file is direct (random). Direct files do not use dynamic allocation. You can have both linked and direct files in FM (table files, for example, are direct).

INDEX FILENAME:

KEY DESCRIPTOR RECORD #:

For each index file associated with your data file, you must supply the logical name of the index (as it appears in PARAM) and the record number in the table file where you'll put the key descriptions (type 6 records). For index file CUSTI1, our type 6 record to describe its key is record #14. For index file CUSTI2, our type 6 record to describe its key is record #15. For index file CUSTI3, the type 6 record that describes its key is record #13. Type 6 (key descriptor) records are usually located at records 13-15 (as ours are). They also point to type 5 (field descriptor) records which contain the fields for these keys. If you set up your keys in this manner, you'll be able to find any record using any key by typing a response to one of the keys (see "Using FM" later in this chapter). For now, fill in the answers as we have. When we discuss type 5 and 6 records, you will put your key descriptions in the right places if you follow our examples.

 # RECORD TYPES:

You should answer with the number of record types. We only have 1, so we enter 1. FM uses the first 2 bytes (status flag) of every record to determine what type of record it is. A positive type causes FM to automatically check this status flag. The status flag tells FM what type the record is, so that FM can select the correct page format when you request that record. Thus, a status of 1 (type 1) would cause format 1 to be used. You can have up to nine record types. If your type is negative, FM does not automatically select a format for your record type -- you have to select the format yourself (using the CHANGE FORMAT key or the FORMAT command in roll-mode) -- and FM does not check the status flags. You *cannot* answer this question with a zero, because a zero in a status flag means a deleted record. For now, answer with a positive one until you are more familiar with FM.

Licensed Material-Property of Data General Corporation

INDICATOR:

This question is asked solely to keep unauthorized people from building table files. You have to answer 31416 before FM will use this table file. Your system manager can change this number, so check with your system manager first.

MAX FIELDS:

You must answer with a value greater than or equal to the largest number of fields shown on any given page including key fields. FM uses this number to allocate work space, and you'll get an error if you use a format with more fields than this maximum. Count the number of fields (input or output values) you'll be displaying, and add a few more to allow changes and additional fields. We will have 9 fields -- CUST, ZIP, ALPHA, NAME, ADDR, CITY, BAL, PHONE, and SEX -- so we enter 10 (just in case we want to add one more field).

DEFAULT FORMAT:

You answer with 1 which is the record number of your first type 4 record, minus 2 (type 4 records start at record #3). This is the relative number of the screen format to be used when you execute FM on your data file. Your type 4 records always start at record #3, but you can have up to 9 type 4 records -- one for each data record type. Since we only have one data record type, we only have one type 4 record, that occurs in record #3. If you have more than one data record type, you'll have more than one type 4 record, and the first format you want FM to use should be one of these type 4 records. Answer with the record number of the type 4 record you want FM to use first, minus 2. FM will first use the format described in that record. FM finds the other formats (type 4 records) using the formula format# + 2 to get to the type 4 record, so keep your type 4 records sequential (see "Type 4 records" later in this chapter).

DEFAULT PAGE:

Answer with the number of the page you want to be initially displayed after a FIND function. We only need 1 page per record, so we enter a 1 here. Most likely, you'd want your first page (#1) to be displayed initially.

When you've finished with the type 1 record in your table file, hit the CHANGE FORMAT key (or type FORMAT as your function in roll-mode) to specify record type 2.

Type 2 Records

Type 2 records are always record #1 in table files. They are a continuation of control records. Figure 5-4 shows our type 2 record for CUST.TB.

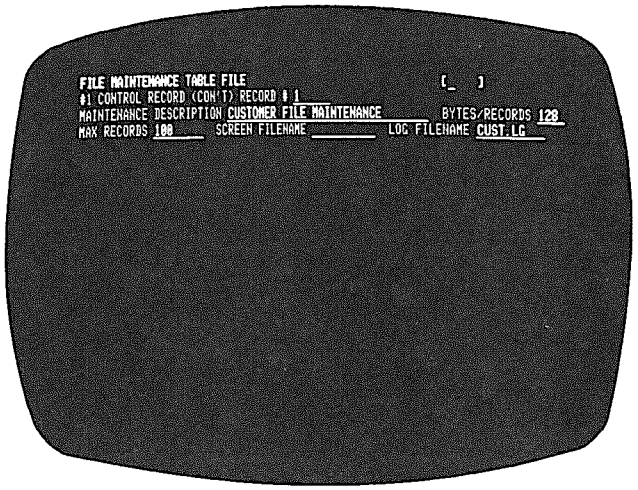

FILE MAINTENANCE TABLE FILE [_]
#1 CONTROL RECORD (CON'T) RECORD # 1
MAINTENANCE DESCRIPTION CUSTOMER FILE MAINTENANCE BYTES/RECORDS 128
MAX RECORDS 100 SCREEN FILENAME LOG FILENAME CUST.LG

Figure 5-4. Type 2 Record

Your type 2 record must be record #1. We answer CUSTOMER FILE MAINTENANCE as our maintenance description; this message will appear whenever we access file CUST using FM (you can write whatever message you want). Bytes per record and maximum number of records must be answered according to what values you assigned your data file when yu cataloged it in a PARAM. Our data file CUST has 128 bytes per record and has 100 records. FM uses these values if FM can't find your data file in PARAM. Remember, however, that if you're using subfiles, you still need to catalog them in PARAM. Otherwise, they will be considered to be physical files.

We left the SCREEN FILENAME field blank because we use the default FM screen. If you want to specify a particular screen to be used, you must enter the screen filename here (screen files are described in Chapter 6). If you use a screen file built by SM, your screen must already by built to contain enough unprotected fields to hold all fields for a given format/page combination. You must also assign a special screen number to your screen. You arrive at the screen number by using this formula:

```
screen# = ((page#)-1*(# of record types) +
  ((format #)-1)
```

Where:

screen# is the number for your screen.

page# is the number of the page for this screen.

format# is the number you entered as the answer to DEFAULT FORMAT: in type 1 records of a table file.

Licensed Material-Property of Data General Corporation

For LOG FILENAME: we answered CUST.LG, which will be our log file maintained by FM. If you want to use a log file, you must CRAND (or INITFILE) it first. You don't have to do anything more to it. The log file automatically receives a copy of all adds, changes, and deletes performed on the data file associated with it. If you leave this field blank, no logging is done by FM. If you establish a log file, its record size must be 18 bytes larger than the data file's record size.

When you're done with your type 2 record, use the CHANGE FORMAT key (or the FORMAT function with roll-mode) to specify record type 3.

Type 3 Records

Figure 5-5 shows a type 3 record which must always be record number 2 in your table file.

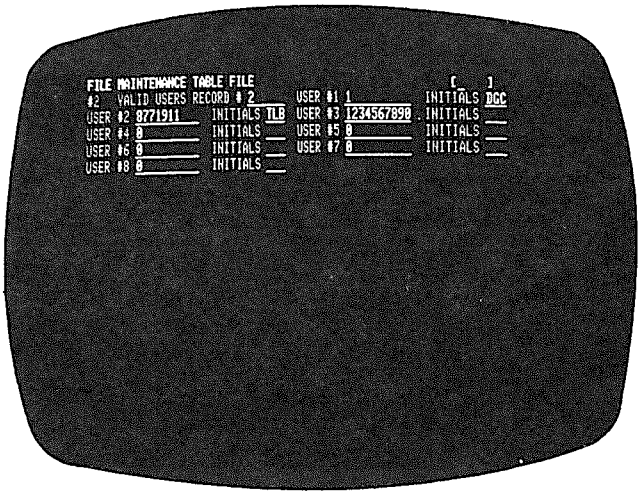

FILE MAINTENANCE TABLE FILE
#2 VALID USERS RECORD # 2
USER #1 1 INITIALS DGC
USER #2 8771911 INITIALS TLB
USER #3 1234567890 INITIALS
USER #4 0 INITIALS
USER #5 0 INITIALS
USER #6 0 INITIALS
USER #7 0 INITIALS
USER #8 0 INITIALS

Figure 5-5. Type 3 Record

Type 3 records maintain a password system for your data file. You can set up to a 10-digit number FM will use to match against a user ID. If you want your file to be public, set the first USER # field to 0, and FM will not check this record. If you use this ID validation, whenever anyone requests this data file using FM, they will have to answer a USER NUMBER: question first, and FM will match the number with the initials you set in record 2 of your table file. If the number does not match with the initials you used when you signed on, you get an error message, and it continues to ask the question until you type in a valid number of IKEY out of the program.

We assign the initials DGC to password 1 for our file -- CUST. If DGC signs on, he/she must supply a 1 for user number. We assigned 8771911 to user TLB, and we assigned a general password 1234567890 to anyone who logs on.

When you're through with this record, you again use the CHANGE FORMAT key (or the FORMAT function for roll-mode), and select record type 4.

Type 4 Records

Type 4 records describe the formats used to display fields. They always start at record 3, and proceed sequentially to 11, allowing up to 9 different format types for 9 different data record types. Each format can have up to 8 pages. You select a page by using the PAGE CHANGE key (or the PAGE function in roll-mode) and you select a different record format using the CHANGE FORMAT key (or the FORMAT function in roll-mode); both are described in "Using FM".

Record 3 (type 4) corresponds to your first data record type; record 4 to your second, etc. The DEFAULT FORMAT field in record 0 (type 1) tells FM which data record type (and its format) should be displayed when you do your first FIND. Our type 4 record for CUST.TB (record 3) is shown in Figure 5-6.

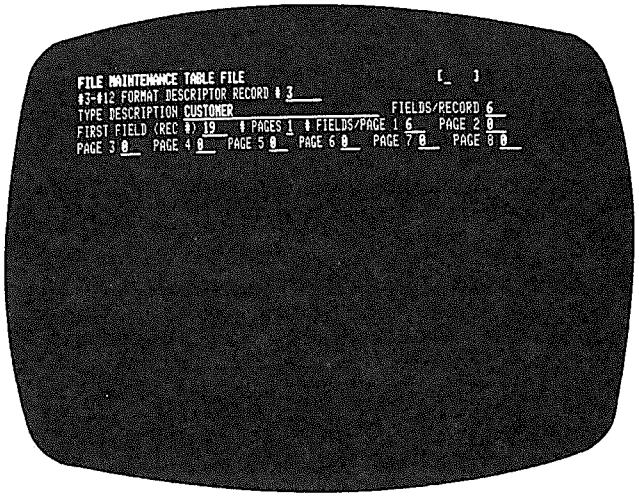

FILE MAINTENANCE TABLE FILE
#3-#12 FORMAT DESCRIPTOR RECORD # 3
TYPE DESCRIPTION CUSTOMER
FIELDS/RECORD 6
FIRST FIELD (REC #) 19 # PAGES 1 # FIELDS/PAGE 1 6 PAGE 2 0
PAGE 3 0 PAGE 4 0 PAGE 5 0 PAGE 6 0 PAGE 7 0 PAGE 8 0

Figure 5-6. Type 4 Record

TYPE DESCRIPTION

Your answer to this field will be displayed on the second line of the screen when you call this record. Our record identifier is CUSTOMER.

Licensed Material-Property of Data General Corporation

FIELDS/RECORD:

Here enter the number of field descriptor records (type 5 records) you will be using. You need one record for each data field you are going to display. Each type 5 record will contain the prompt you want and the size and data type of the field you plan to display. We entered a 6 for the following data fields: NAME, ADDR, CITY, BALANCE, PHONE, and SEX.

FIRST FIELD (REC #):

We answered 19 because record #19 is where we keep our first data field (NAME) information in field descriptor record (type 5) number 19. Type 5 records start at 16 and can go up to 400, but we use 16, 17, and 18 for index entry fields. Starting at 19, our type 5 records describe each of our 6 data fields up to record 24. FM expects to find the next consecutive field in the next consecutive record.

 # PAGES:

Answer with the number of pages required for this type of record format. Pages are separate screens, and you can specify a page with the CHANGE PAGE key (or the PAGE function in roll-mode). We answer 1 because we only need one page.

 # FIELDS/PAGE 1;... PAGE 8

For each page of your record format, you specify the number of fields per page. This tells FM how many consecutive field descriptor records (type 5 records) should be counted as one page. When FM displays the first page, it uses the first 6 records starting with 19 (19-24). If we needed subsequent pages, we'd put the second page's fields in records immediately following the first page's (starting at record 25). Your answer to PAGE 2: would tell FM how many records (fields) to count as page 2.

Type 5 Records

Type 5 records describe the individual fields used for each data value input or output to the CRT screen. These field descriptor records must exist for all key and data fields and for any field to be displayed and maintained. Starting at record 16, you must use consecutive records (1 field per record) in the order you want the fields displayed, and each page's fields must follow consecutively in the page order you want.

Our records 16, 17, and 18 are describe key fields (index entries, one of which the user must key in to find a record). Figure 5-7 shows record 16.

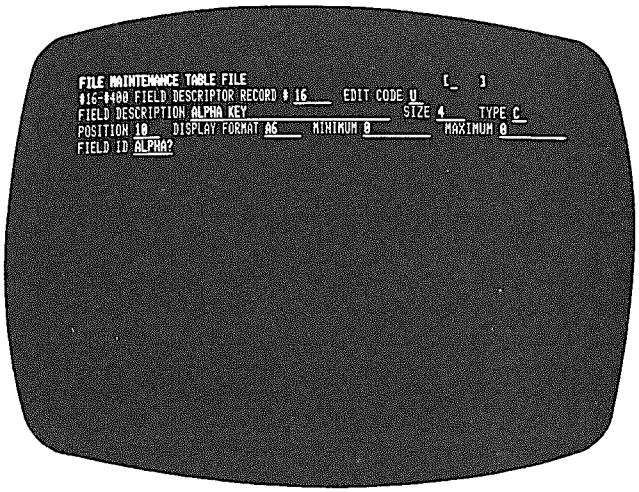


Figure 5-7. Type 5 Record

EDIT CODE:

A U stands for an unedited (alphanumeric) field and an N stands for an edited numeric field. In addition to U or N, you can have an R indicating that the data in the field should be right-justified (by default, it is left-justified).

FIELD DESCRIPTION

You type the prompt you want for this field. Our field for record 16 is the ALPHA key; this prompt will appear before the input field.

SIZE:

Answer with the size in bytes of this field's data in the data record. Our ALPHA key is 4 bytes long.

TYPE:

You answer with one (or two) character codes indicating the format of the field in the data record. These codes are:

C crammed field (3 characters per 2 bytes).
S string field.
I integer (binary) field, single precision (2 bytes).
D double integer (binary) field (4 bytes).
B binary field, usually 1 or 3 bytes.
N digit subfield (part of a larger number).
F bit flag field.

Some combinations are possible, like CS and BF (or BN), etc.

Licensed Material-Property of Data General Corporation

POSITION:

You answer with the starting byte location in the data record (relative to 0) of the data associated with this field.

DISPLAY FORMAT:

You answer with a format code appropriate for this field (how it should appear on the terminal screen); they are similar to the PRINT USING formats:

F *w.d* display binary or integer field right-justified with a maximum of *w* digits with *d* digits to the right of the decimal point.

A *w* display string or crammed field with a maximum of *w* characters.

B *w.d* display bit flag field with *w* spaces setting bit corresponding to 2 ↑ *d* (0 <= *d* <= 31); *w* must always equal 1.

N *w.d* display *w* digit subfield where the rightmost digit is *d* digits from the right of the major field. For example, for 123456, N3.1 gives you 345.

MINIMUM: and *MAXIMUM:*

If you define your field as numeric edited (N), you can set the smallest and largest values allowable for the field. If you leave a 0 in both, and a numeric is input, you'll get an error. These restrictions are used primarily for key entries, to inhibit you from asking for a key that is an out-of-range number. String fields do not have these restrictions, so you can leave zeros in string fields.

FIELD ID:

Answer with the prompt you want to use if this file will be used with roll-mode File Maintenance. This prompt can only be 6 characters long, and appears instead of the FIELD DESCRIPTION answer (which appears on the CRT for regular FM).

You should step through the previously build CUST.TB file (using FIND NEXT) to look at the rest of the field descriptor records.

Type 6 Records

Type 6 records are records 13 to 15 in your table file (1 for each of 3 index files allowed). You can make any given key a composite of up to 3 fields in the data record with 3 separate fields to be displayed, and each record type (format) can have 3 key fields, as well as each key having 3 fields. The combined length of all fields for a single key must not exceed 30 bytes.

Figure 5-8 shows our type 6 record (record 13 for our ALPHA key).

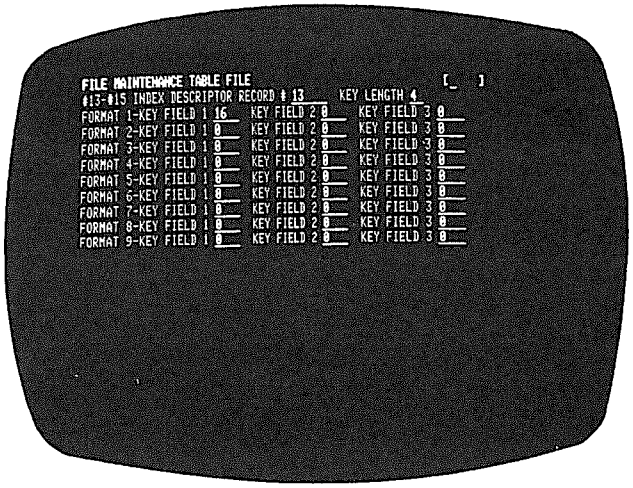

FILE MAINTENANCE TABLE FILE
#13=015 INDEX DESCRIPTOR RECORD # 13
KEY LENGTH 4
FORMAT 1-KEY FIELD 1 16 KEY FIELD 2 0 KEY FIELD 3 0
FORMAT 2-KEY FIELD 1 0 KEY FIELD 2 0 KEY FIELD 3 0
FORMAT 3-KEY FIELD 1 0 KEY FIELD 2 0 KEY FIELD 3 0
FORMAT 4-KEY FIELD 1 0 KEY FIELD 2 0 KEY FIELD 3 0
FORMAT 5-KEY FIELD 1 0 KEY FIELD 2 0 KEY FIELD 3 0
FORMAT 6-KEY FIELD 1 0 KEY FIELD 2 0 KEY FIELD 3 0
FORMAT 7-KEY FIELD 1 0 KEY FIELD 2 0 KEY FIELD 3 0
FORMAT 8-KEY FIELD 1 0 KEY FIELD 2 0 KEY FIELD 3 0
FORMAT 9-KEY FIELD 1 0 KEY FIELD 2 0 KEY FIELD 3 0

Figure 5-8. Type 6 Record.

The KEY LENGTH field should contain the maximum size in bytes of the key. For each format (record type), you should answer KEY FIELD 1, KEY FIELD 2, and KEY FIELD 3 with the record numbers of the field descriptor (type 5) records that describe the field. A zero indicates that the field is not used. You can make each record type (format) have different keys (e.g., 3 keys: one in record 13, one in 14 and one in 15), and each key can be composed of 1 to 3 different fields.

Our ALPHA key is 4 bytes long, and it only uses one field, and that field is described in record 16 (a type 5 record).

When you're finished with your table file, you are ready to use your data file with FM. There are other utilities described in this chapter (MOVETABREC and TABBUILD) which help you duplicate table file records and build field definitions for recurring fields. The utility FMTABPRINT prints your completed table file at your terminal or line printer.

Using FM

Regular FM can run on DG 6053 terminals; if you have a DG 6012, teletype, or other terminal, the FM program will choose the roll-mode version. You can type FM in the BASIC CLI, or you can CHAIN, SWAP, or RUN "FM". The program begins by asking for a filename. If you're creating a table file (see previous section), you have to type FM.TB; for any data file for which you've already set up a table file, just type the name of the data file. For our examples, type CUST.

Licensed Material-Property of Data General Corporation

FM will ask you for a user number. User validation codes are set up in record 2 (type 3) of the table file (CUST.TB). The user number question is, however, optional. Note that the user number is not echoed when you type it.

Roll-mode FM is discussed after this section.

Using Function Keys

Your FM function keys are the top row of keys on a 6053 terminal. You should use the template provided in Figure 5-1 and Table 5-2. Whenever you get an FM error message, press the CLEAR ERROR key. You can only use the other function keys when the cursor is put in the square brackets.

FIND

When you press the FIND key, the system expects you to give a record number or a key entry, allowing a maximum of three key entries. You can't find record 0 in a linked-available-record file. FIND finds a record from a key using the ISAM method described in Chapter 4. Figure 5-9 demonstrates a FIND on file CUST using carriage returns to get to the ALPHA KEY field and typing:

ALPHA KEY OR)

FIND NEXT

From any record (after a FIND) you can find the next record in sequence based on either record number (if you used record number in a FIND) or the last key used. For example, if the last FIND used the ALPHA KEY in CUST, then FIND NEXT would find the next alpha key in sequence. You can step through data files using any key in sequence, but when advancing sequentially through a direct file by record number, you'll be stopped by a deleted record. You must use FIND to find a higher record.

CHANGE

You can change any field's data (except the record number) by using CHANGE and typing the new data as you are positioned to the field. You can use the carriage return key to move forward without disturbing data in fields and to finish the editing session, but if you change anything on the screen, it will be stored when you complete the function (carriage return at the last field, or the HOME key). You must retype the entire field that you're changing. If you want to get out of a change function and preserve what you had, use the IKEY (the interrupt key, usually ESC).

ADD

If your file is a linked-available-file (dynamic record allocation) ADD finds the next available record (either a deleted record or the next one in sequence). If your file is direct (random), you have to specify a record number. Note that if the # RECORD TYPES field in record 0 (type 1) of the table file is zero, FM will not check the record status (first 2 bytes) to see if it is an active record. An ADD in this case would overwrite an active record if you specified an active record. A direct (random) file would also not have status flags, so their records could be overwritten. A good practice is to set the # RECORD TYPES field to at least 1.

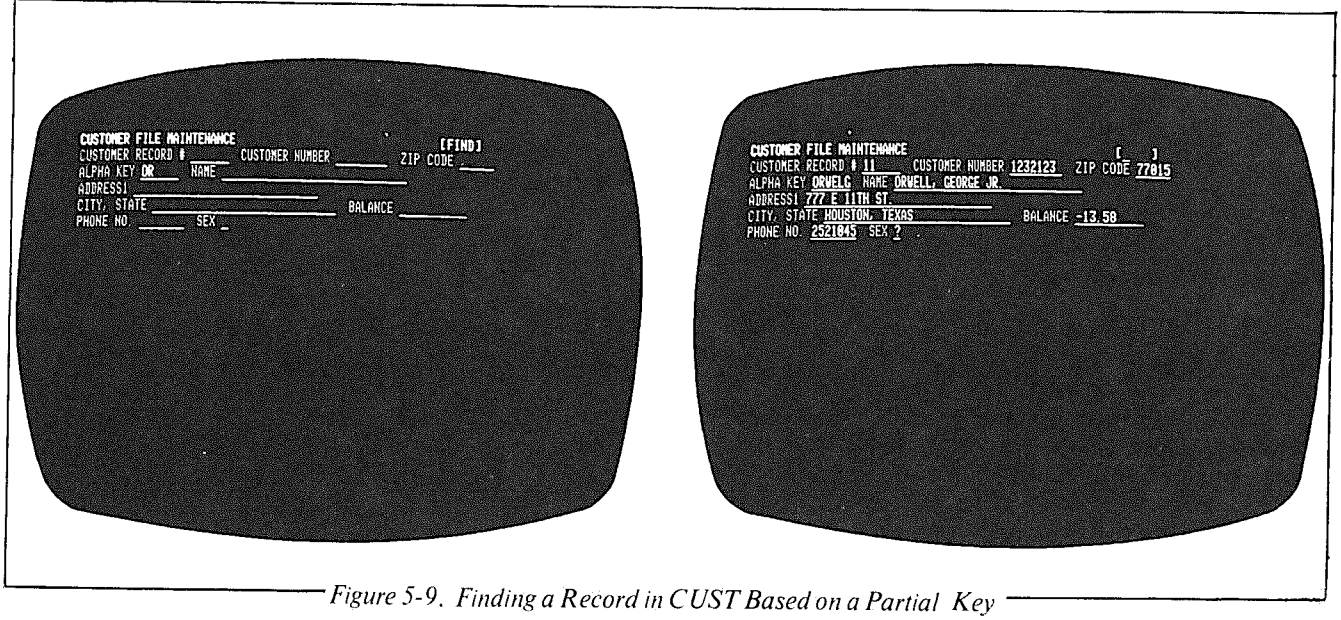


Figure 5-9. Finding a Record in CUST Based on a Partial Key

Licensed Material-Property of Data General Corporation

When you are adding new data, all fields are checked for range and data type errors. Any error aborts the ADD and gives you an error message, which you can only clear with the CLEAR ERROR key. When adding keys, you'll get an error message if you try to add a duplicate key to an index file that won't allow duplicate keys.

DELETE

A DELETE deletes the record currently displayed from the data file. You have to FIND the record first. The status flag (first 2 bytes) of the record is set to 0.

CLEAR ERROR

Whenever you get an FM error, use this key. If the error occurred in one of the fields, you would be put back into that field. Otherwise CLEAR ERROR would return you to the function brackets.

CHANGE FORMAT

You must have a file with multiple record types for this to work. You can change the format using this key if you're adding a new record not of the type currently displayed (or changing record types in a file with no record status), and it is used to change record types to create table file records (see last section). By default, the format corresponding to the value of the record's status (first 2 bytes) is displayed. You can change it by hitting CHANGE FORMAT and typing the type number in place of the record number.

CHANGE PAGE

CHANGE PAGE displays another page of current record (a page is a screen display). You hit CHANGE PAGE and type the number of the page in place of the record number. If this is a valid page number, the new page will appear.

The record number and any key fields are retained between pages. If you ADD or CHANGE any page's field, it changes the data in the record (which means that you can CHANGE or overwrite a record from any page of it).

STOP

STOP terminates FM and returns you to keyboard mode, or to whichever program swapped to it.

Roll-mode FM

Roll-mode FM will work on the same files that regular FM sets up. The only difference is that a screen form is not displayed. A FUNCTION? query occurs after every function execution and after every error. You type the command corresponding to the function you want; all the functions are summarized in Table 5-2 and described in detail in the previous section. Two functions unique to roll-mode are AUTO and DISPLAY.

AUTO turns on the automatic display to show the abbreviated fields; DISPLAY shows the current record's fields (or page). CHANGE displays one field at a time for you to change. See Example 5-1.

Example 5-1

```
FUNCTION? CHANGE )
REC # 1 CUST:1/ )   ZIP:   30330/ )   ALPHA: TEXT1 / )
NAME: TEST ACCOUNT/ )   ADDR: 123 E. SOMEPLACE/ )
REQUEST NOT COMPLETED
FUNCTION? )
REC # 2
FUNCTION? )
REC # 3
FUNCTION? )
```


Licensed Material-Property of Data General Corporation

You enter your change after the slash (/), press carriage return to get to the next field, or you hit the TAB key and the FIELD? question appears, as shown in Example 5-2.

Printing a Table File List

The utility FMTABPRINT provides a convenient means of examining any table file used by FM. You can RUN, CHAIN or SWAP to FMTABPRINT, and it will ask you for the filename of the table file. If you execute FMTABPRINT from the BASIC CLI, use this form:

FMTABPRINT *[/L]* tablefilename

The /L switch sends the output to the line printer. If you RUN FMTABPRINT, the system asks you if you want the output sent to the line printer.

FMTABPRINT looks for a .DS file (filename with a .DS extension with the same name as the table file). This is assumed to be the comment file for your table file. If you don't have a .DS file, the table file is printed with default information only. The .DS file contains special lines that take the form:

\field

where field is the name of a field in the field descriptor records. When FMTABPRINT sees one of these lines, it puts the text following it (up to the next special line) after the corresponding information for the field. Any text in the .DS file before the first field line will be printed as heading information. Make sure you keep your field lines in the .DS file in the same order as the fields appear in an FMTABPRINT printout. See Example 5-3 for an FMTABPRINT printout of CUST.TB.

Licensed Material-Property of Data General Corporation

Example 5-2

```
                                    FUNCTION? CHANGE )
                                    REC # 1  CUST: 1 / tab    FIELD? ALPHA )
                                    ALPHA: TEST1 / ikey
                                    REQUEST NOT COMPLETED
                                    FUNCTION? ADD )
                                    REC # 1 )   CUST: 2 )   ZIP:
```

An interrupt aborts a CHANGE or ADD session. Note that ADD does not display the old values.

Example 5-3

```
FILENAME: CUST                  LINKED

TABLE FILE: CUST.TB
RECORD LENGTH: 128              MAXIMUM NUMBER OF RECORDS: 100
CUSTOMER FILE MAINTENANCE
CUSTOMER RECORD  ( TYPE 1 )
REC  NAME   SEQ       DESCRIPTION                  SIZE TYPE POS. FORMAT EDIT
INDEX FILE: CUSTI1              KEY LENGTH: 4
 17  CUST     1  CUSTOMER NUMBER                     4   D     2  D8.0    N          0     99999999

INDEX FILE: CUSTI2              KEY LENGTH: 4
 18  ZIP      1  ZIP CODE                           4   D     6  D5.0    N          0       99999

INDEX FILE: CUSTI3              KEY LENGTH: 4
 16  ALPHA?   1  ALPHA KEY                          4   C    10  A6      U

 19  NAME     1  NAME                              20   C    14  A30     U
 20  ADDR     2  ADDRESS1                          30   S    34  A30     U
 21  CITY     3  CITY, STATE                       20   C    64  A30     U
 22  BAL      4  BALANCE                            4   D    84  D10.2   U
 23  PHONE    5  PHONE NO.                          3   B    88  D7.0    N          0     9999999
 24  SEX      6  SEX                                1   S    92  A1      U
```

Licensed Material-Property of Data General Corporation

Printing an FM Data File

The utility FMPRINT prints data records as if you were using NEXTs. You can select first and last record numbers, or first and last key values. You can also specify your output file.

You can RUN, SWAP, or CHAIN FMPRINT, or execute FMPRINT from the BASIC CLI. To use FMPRINT, you must follow the same log-on sequence you would follow to use the file with FM.

You can optionally choose to put a form feed between records. You can also specify an output format: spaced, packed, or full. Examples 5-4, 5-5, and 5-6 show spaced, packed, and full formats, respectively.

Printing a Log File

FMLOG is a utility program that will print your log file maintained by FM. FMLOG works just like FMPRINT -- you have the same choice of output file and formats. In addition, FMLOG also asks (after outputting the file) if you want to clear the file (delete the records). Log files are automatically maintained for you. Example 5-7 shows sample output from FMLOG.

Copy Table File Records

The utility MOVETABREC duplicates table file records within a given table file. You will find this useful for defining multiple pages and/or formats where the same fields appear on more than one page and/or format. Example 5-8 shows a sample run of MOVETABREC.

Quick Definition of Arrays

The utility program TABBUILD provides a quick method of creating field descriptor records for recurring fields of the same type. Usually these fields are part of an array occurring in a data record type. TABBUILD requests the same fields (FORMAT:, TYPE:, etc.) as FM.TB asks to complete a field descriptor record. Also, TABBUILD asks for a starting record number in the table file and a record increment. Thus you can leave space between table records which will be filled in with associated fields when building a data structure. Likewise, a starting position and a position increment within the data record is requested. To make it easier to identify recurring fields, an element number is automatically appended to the DESCRIPTION and ID.

As an example, if a second record type were added to the CUST file containing the number of service calls received per month, you could use TABBUILD to add these fields to CUST.TB. Example 5-9 shows the use the TABBUILD.

Licensed Material-Property of Data General Corporation

093-000137-03

Example 5-4

```
REC#: 1       CUST: 1          ZIP: 30330  ALPHA?: TEST1   NAME: TEST ACCOUNT
ADDR: 123 E. SOMEPLACE                CITY: ATLANTA,GA                        BAL: 123.45      PHONE: 2521045  SEX: M


REC#: 2       CUST: 50         ZIP: 11111  ALPHA?: TEST2   NAME: TEST CASE 21
ADDR: 123 EAST 23 STREET              CITY: CITY 2                            BAL: 100.00      PHONE: 4512345  SEX: F


REC#: 1       CUST: 1       ZIP: 30330       ALPHA?: TEST1       NAME: TEST ACCOUNT       ADDR: 123 E. SOMEPLACE
CITY: ATLANTA,GA       BAL: 123.45       PHONE: 2521045       SEX: M

REC#: 2       CUST: 50       ZIP: 11111       ALPHA?: TEST2       NAME: TEST CASE 21       ADDR: 123 EAST 23 STREET
CITY: CITY 2       BAL: 100.00       PHONE: 4512345       SEX: F
```

Example 5-5

```
REC#: 1  CUST: 1  ZIP: 30330  ALPHA?: TEST1  NAME: TEST ACCOUNT  ADDR: 123 E. SOMEPLACE  CITY: ATLANTA,GA  BAL: 123.45
PHONE: 2521045  SEX: M

REC#: 2  CUST: 50  ZIP: 11111  ALPHA?: TEST2  NAME: TEST CASE 21  ADDR: 123 EAST 23 STREET  CITY: CITY 2  BAL: 100.00
PHONE: 4512345  SEX: F


REC#: 1  CUST: 1  ZIP: 30330  ALPHA?: TEST1  NAME: TEST ACCOUNT  ADDR: 123 E. SOMEPLACE  CITY: ATLANTA,GA  BAL: 123.45
PHONE: 2521045  SEX: M

REC#: 2  CUST: 50  ZIP: 11111  ALPHA?: TEST2  NAME: TEST CASE 21  ADDR: 123 EAST 23 STREET  CITY: CITY 2  BAL: 100.00
PHONE: 4512345  SEX: F
```

Example 5-6

```
REC#: 1       CUST: 1       ZIP: 30330       ALPHA?: TEST1       NAME: TEST ACCOUNT       ADDR: 123 E. SOMEPLACE
CITY: ATLANTA,GA       BAL: 123.45       PHONE: 2521045       SEX: M

REC#: 2       CUST: 50       ZIP: 11111       ALPHA?: TEST2       NAME: TEST CASE 21       ADDR: 123 EAST 23 STREET
CITY: CITY 2       BAL: 100.00       PHONE: 4512345       SEX: F


REC#: 1       CUST: 1          ZIP: 30330  ALPHA?: TEST1   NAME: TEST ACCOUNT
ADDR: 123 E. SOMEPLACE                CITY: ATLANTA,GA                        BAL: 123.45      PHONE: 2521045  SEX: M

REC#: 2       CUST: 50         ZIP: 11111  ALPHA?: TEST2   NAME: TEST CASE 21
ADDR: 123 EAST 23 STREET              CITY: CITY 2                            BAL: 100.00      PHONE: 4512345  SEX: F
```

Licensed Material-Property of Data General Corporation

Example 5-7

```
RECORD ADDED
 BY AADGC6 ON  5/17/78 AT 19:41:08
REC#: 29      CUST: 7         ZIP: 19085  ALPHA?: BOND    NAME: JAMES T. BOND
ADDR: NO. 7 WALKINSTEP              CITY: LONDON, ENGLAND, G.B.          BAL: 4.73        PHONE: 7         SEX: M

RECORD BEFORE CHANGE
 BY AADGC6 ON  5/17/78 AT 19:44:23
REC#: 29      CUST: 7         ZIP: 19085  ALPHA?: BOND    NAME: JAMES T. BOND
ADDR: NO. 7 WALKINSTEP              CITY: LONDON, ENGLAND, G.B.          BAL: 4.73        PHONE: 7         SEX: M

RECORD AFTER CHANGE
 BY AADGC6 ON  5/17/78 AT 19:44:41
REC#: 29      CUST: 7         ZIP: 19085  ALPHA?: BOND    NAME: JAMES T. BOND
ADDR: NO. 7 WALKINSTEP              CITY: LONDON, ENGLAND, G.B.          BAL: 4.73        PHONE: 7         SEX: M

RECORD BEFORE CHANGE
 BY AADGC6 ON  5/17/78 AT 19:44:43
REC#: 29      CUST: 7         ZIP: 19085  ALPHA?: BOND    NAME: JAMES T. BOND
ADDR: NO. 7 WALKINSTEP              CITY: LONDON, ENGLAND, G.B.          BAL: 4.73        PHONE: 7         SEX: M

RECORD AFTER CHANGE
 BY AADGC6 ON  5/17/78 AT 19:45:39
REC#: 29      CUST: 7         ZIP: 19085  ALPHA?: BOND    NAME: JAMES T. BOND
ADDR: NO. 7 ST. JAMES, BOND ROW     CITY: LONDON, ENGLAND, G.B.          BAL: 47434.00    PHONE: 1000007   SEX: M

RECORD ADDED
 BY AADGC6 ON  5/18/78 AT    30:32
REC#: 30      CUST: 0         ZIP: 0      ALPHA?:         NAME:
ADDR:                               CITY:                                BAL: 0.00        PHONE: 0         SEX:

RECORD DELETED
 BY AADGC6 ON  5/18/78 AT    30:47
REC#: 30      CUST: 0         ZIP: 0      ALPHA?:         NAME:
ADDR:                               CITY:                                BAL: 0.00        PHONE: 0         SEX:

RECORD BEFORE CHANGE
 BY AADGC6 ON  5/19/78 AT  4:29:30
REC#: 11      CUST: 1232123   ZIP: 77015  ALPHA?: ORWELG  NAME: ORWELL, GEORGE JR.
ADDR: 777 E 11TH ST.                CITY: HOUSTON, TEXAS                 BAL: -13.58      PHONE: 2521045   SEX:

RECORD AFTER CHANGE
 BY AADGC6 ON  5/19/78 AT  4:29:41
REC#: 11      CUST: 1232123   ZIP: 77015  ALPHA?: ORWELG  NAME: ORWELL, GEORGE JR.
ADDR: 777 E 11TH ST.                CITY: HOUSTON, TEXAS                 BAL: -13.58      PHONE: 2521045   SEX: ?
```

Licensed Material-Property of Data General Corporation

093-000137-03

Licensed Material-Property of Data General Corporation

Example 5-8

```
*RUN "MOVETABREC" )

MOVE RECORDS IN A TABLE FILE
TABLE FILE NAME: CUST.TB )
SOURCE RECORD NUMBER: 16 )
DESTINATION RECORD NUMBER: 36 )
NUMBER OF RECORDS: 6 )
VERIFY? YES )
16    ALPHA KEY
17    CUSTOMER NUMBER
18    ZIP CODE
19    NAME
20    ADDRESS1
21    CITY, STATE

SOURCE RECORD NUMBER: ikey
*
```

Example 5-9

```
*RUN "TABBUILD" )
TABLE FILE NAME: CUST.TB )
FORMAT: D4.0 )
TYPE: I )
EDIT CODE: N )
MINIMUM: 0 )   MAXIMUM: 9999 )
RECORD INC: 1 )   POSITION INC: 1 )
STARTING TABLE RECORD: 45 )
STARTING POSITION: 4 )
NUMBER OF RECORDS TO BE CREATED: 12 )
FIRST NUMBER TO APPEND TO FIELD NAMES: 1 )
DESCRIPTION: CALLS FOR THE MONTH )

ID: CPM )

45   4    CALLS FOR MONTH 01   CPM1
46   5    CALLS FOR MONTH 02   CPM2
47   6    CALLS FOR MONTH 03   CPM3
48   7    CALLS FOR MONTH 04   CPM4
49   8    CALLS FOR MONTH 05   CPM5
50   9    CALLS FOR MONTH 06   CPM6
51   10   CALLS FOR MONTH 07   CPM7
52   11   CALLS FOR MONTH 08   CPM8
53   12   CALLS FOR MONTH 09   CPM9
54   13   CALLS FOR MONTH 10   CPM10
55   14   CALLS FOR MONTH 11   CPM11
56   15   CALLS FOR MONTH 12   CPM12

STARTING TABLE RECORD: ikey
*
```

End of Chapter

Chapter 6
Screen Handling

Business BASIC has a utility program called SM (Screen Maintenance) which allows you to define screen formats for input and output, and the subroutines FORM.SL, UNFORM.SL, and PROTFORM.SL which use screen files in your programs. In this chapter we show you how to define a screen using SM, and then we show sample uses of the subroutines. FORM.SL, UNFORM.SL, and PROTFORM.SL are individually described in Chapter 7.

SM creates and stores screen information in screen files, using characters and tables to define fields. SM only works for 6052-3 terminals that have special unmarked function keys. We supply a template for you to use with these terminals; in case you don't have a template, we also supply a paper *template* in this chapter that indicates what each function key does.

A typical screen has high intensity fields, low intensity fields, blinking fields, etc. so that the screen looka like a form for data entry or retrieval and the terminal operator just fills in the blanks. High intensity fields are blanks you fill in, and low intensity fields serve as prompts or output. Figure 6-1 shows a simple screen where NAME, ADDRESS, CITY, STATE, ZIP, and OUTPUT are low intensity fields, and the underscored areas are high intensity fields. From now on we call low intensity fields, protected fields, and high intensity fields are called unprotected fields. Protected fields do not actually stop you from typing into areas on the screen, but they do provide a form that positions the typist to the correct locations. Your program that uses the FORM.SL, UNFORM.SL, and PROTFORM.SL subroutines can display the screen, solicit input from the terminal operator, and wait until the operator hits a function key before using the data input. This way the operator can correct his/her input before sending it to the program.

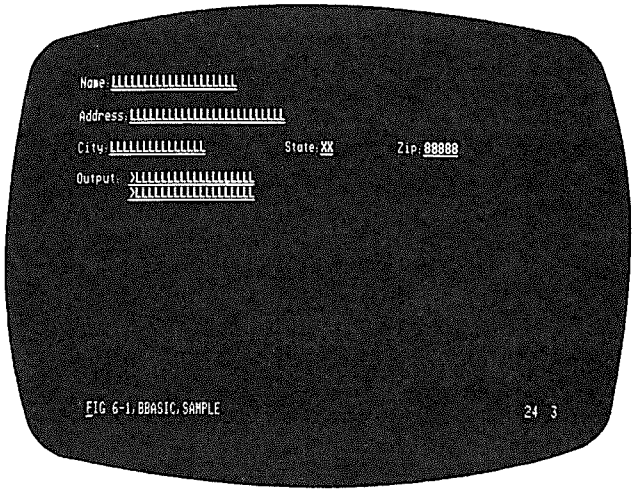


Figure 6-1. A Sample Screen in a Screen File

Defining Your Screen

You can execute SM by typing:

```
*RUN "SM" )
*CHAIN "SM" )
*SWAP "SM" )
!SM )        (from the BASIC CLI)
```


Licensed Material-Property of Data General Corporation

The entire screen blanks out, and this message appears at the top:

DG SCREEN MAINTENANCE REV x.xx mm/dd/yy

SCREEN FILENAME:

As a response, type the name of the screen file you are creating. It can be any name, but by convention we use a filename with a .Sn extension, where n is the terminal type (6 for 6052 or 6053).

SCREEN FILENAME: SCREEN.S6)

The screen blanks out again, but this time you'll see two right angle brackets at the bottom left-hand side of the screen, and some numbers on the bottom right-hand side. The two right angle brackets (>>) are the prompt for SM's special command line. You enter all your SM commands on this line.

You can now use only function keys, character and digit keys, the space key, and the shift key. You can't use IKEY, carriage return, or RUBOUT (DELETE). You can, however, use the cursor positioning keys. The function key template is shown in Figure 6-2. Place them over the function keys on your terminal. Here's a quick summary of each key:

DUP TO EOL (F1):

Duplicates the previous line to the end of the current line. The current line will be exactly like the previous line.

INSERT CHAR (Shift-F1):

Creates a space at the cursor that allows you to insert a character.

PROTECT TO EOL (F2):

After a *Start Protect,* this protects (low intensity) all positions from the cursor to the end of the line.

DELETE CHAR (Shift-F2):

Deletes a character at the cursor.

SCALE TO EOL (F3):

From the cursor, the line is filled with repeating sequences of "1234567890123..." to the end of the line, which is useful as a scale. The cursor moves back to the beginning of the line so that you can type over the scale.

INSERT LINE (Shift-F3):

A new blank line is inserted before the current line.

ENTER CMND (F4):
Positions the cursor to the command line so that you can type one of the SM commands.

DELETE LINE (Shift-F4):

Deletes the current line.

END PROTECT (F5):

Starts an unprotected field or ends a protected one. You should now type into the unprotected field, the special character for the data type you expect on input. The special characters follow this description of the function keys.

START PROTECT (F6):

Starts a protected field, or ends an unprotected one. You can type all sorts of characters in a protected field to be prompts for input, but no input is allowed in a protected field.

TAB TO NEXT (F9):

Moves the cursor to the next unprotected field.

VERIFY SCREEN (F10):

Displays whatever was last displayed on the screen. You can hit this key to see what you have on your screen. After a DISPLAY command, this key only displays protected fields.

PROCESS CMND (F11):

Whenever you issue a command on the command line (if you hit F4 you are positioned there) like READ, WRITE, PRINT, DISPLAY, and STOP, you must hit this key to process the command.

Licensed Material-Property of Data General Corporation

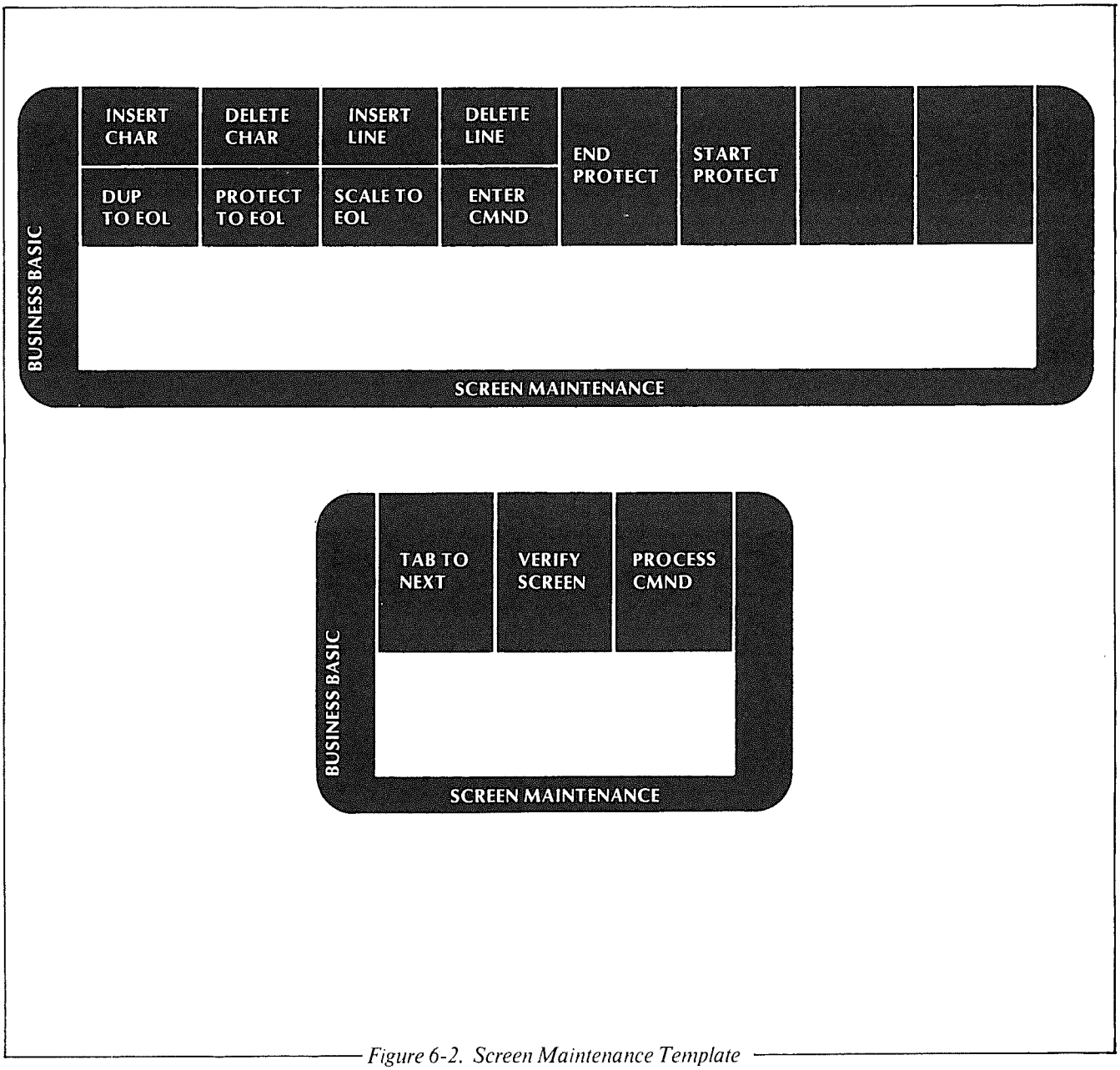


Figure 6-2. Screen Maintenance Template

Licensed Material-Property of Data General Corporation

Special Characters for Formatted Input/Output

You type these characters into your protected and unprotected fields to define formats that only allow certain data types for input/output.

When you fill a protected field with the character 9, it assures that only numeric data can be output to the field. When you fill an unprotected field with the character 8, it assures that only numeric data is allowed as input to the field. A negative in the first position of either field allows negative input/output, and a decimal point in a location allows a decimal point to be input at that location and outputs a decimal point in that location. A floating decimal (:) allows you to type a decimal point anywhere within the input (unprotected) field. A floating decimal point in an output (protected or unprotected) field allows you to output a decimal anywhere in the field.

Your data type is alphanumeric if it is not strictly numeric, decimal, or signed numeric. X/Y allows an alphanumeric field, L/M allows lowercase alphabetic characters (special case of X/Y), and C/D allows a crammed field only -- input is crammed after it is typed, and output is uncrammed before it is printed. The right-justify character (>) justifies any output to the right of a field. Use these formats if you are going to use the FORM.SL subroutine. FORM.SL returns the variable E with a value of 128 if the input is not successful, so you can issue an error message (using UNFORM.SL) to have the operator rekey the screen input.

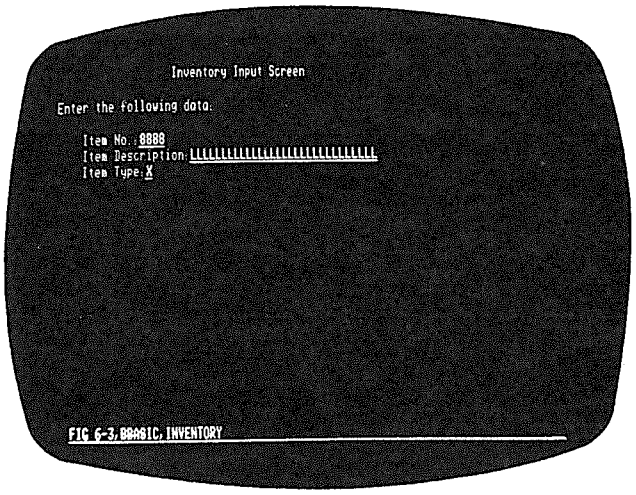
Table 6-1. Special Characters

Character	Field	Data type
8 9	unprotected protected	numeric (fill).
-	both	at the beginning of a numeric field, allows negative values.
.	both	fixed decimal location in a numeric field.
:	both	relative location for a floating decimal point.
X Y	unprotected protected	alphanumeric field (fill).
L M	unprotected protected	alphanumeric field following lowercase characters (fill).
C D	unprotected protected	crammed field (fill).
>	both	right-justified field, put only in first column of field.

Licensed Material-Property of Data General Corporation

First--Key Your Screen:

To move the cursor to the top of the screen, use the HOME key. The cursor positioning keys move the cursor anywhere on the screen. Now you use your function keys. see the printout in Figure 6-3, and try to duplicate it: from the HOME position, hit the START PROTECT (F6) key, and space over to position 19 (you're at position 1). Type INVENTORY INPUT SCREEN in upper- or lowercase. Then hit PROTECT TO EOL (protect to end of line -F2) key, because you should protect *all* unused portions of the screen. The cursor is now on the second line, and since *protect* mode is still on, you can hit PROTECT TO EOL key again. You should type the third line's prompt, followed by another PROTECT TO EOL, another one for the next line, and finally key spaces and ITEM NO.:. Now you're ready for an unprotected field.




```
PROJECT: FIG 6-3

SYSTEM: BBASIC                              PROGRAM: INVENTORY

SCREEN ID: SCREEN.S6[0]     LAST MODIFIED BY: AADGC6 AT 06:45:26 ON 05/16/78

          1         2         3         4         5         6         7         8
  12345678901234567890123456789012345678901234567890123456789012345678901234567890
 .--------------------------------------------------------------------------------.
 1:                  Inventory Input Screen                                        : 1
 2:                                                                                : 2
 3:Enter the following data:                                                       : 3
 4:                                                                                : 4
 5:    Item No.:8888                                                               : 5
 6:    Item Description:LLLLLLLLLLLLLLLLLLLLLLLLLL                                 : 6
 7:    Item Type:X                                                                 : 7
 8:                                                                                : 8
 9:                                                                                : 9
10:                                                                                :10
11:                                                                                :11
12:                                                                                :12
13:                                                                                :13
14:                                                                                :14
15:                                                                                :15
16:                                                                                :16
17:                                                                                :17
18:                                                                                :18
19:                                                                                :19
20:                                                                                :20
21:                                                                                :21
22:                                                                                :22
23:                                                                                :23
  :--------------------------------------------------------------------------------:
  12345678901234567890123456789012345678901234567890123456789012345678901234567890
```

Figure 6-3. Screen for the Sample Program in Example 6-3

Licensed Material-Property of Data General Corporation

Start by hitting the END PROTECT key (F5). Now type 8 four times then hit START PROTECT again, followed by a PROTECT TO EOL. Key ITEM DESCRIPTION:, then type Ls or Xs. (We used Ls to allow lowercase characters, too.) Be sure to protect all unused areas.

If you key this screen in Figure 6-3 exactly, you'll be able to run the complete program we supply in this chapter.

If you make mistakes, use the function keys INSERT CHAR (Shift-F1), DELETE CHAR (Shift-F2), INSERT LINE (Shift F3) and DELETE LINE (Shift F4). If you use cursor positioning to retype lines, make sure you always retype entire fields, not just characters in any one field. You'll usually have to retype the entire line. To redisplay what you've just typed, use the VERIFY SCREEN (F10) key. If you have difficulty judging column positions, use the SCALE TO EOL (F3) key, which will print a scale for you to type over (make sure you type over the whole scale). The TAB TO NEXT (F9) key will move the cursor to the next unprotected field.

Now, WRITE Your Screen:

When you've got your screen the way you want it, hit the ENTER CMND (F4) key. This will move the cursor to the command line at the bottom left-hand side of the screen. You should now type WRITE 0 project-name, system-name, program-name, or just WRITE 0, and execute the command using the PROCESS CMND (F11) key. Your project name, system name, and program name are useful as headings when you PRINT the screen on the line printer, but they are optional. Now 0 is your screen number in your screen file, yourname.Sn (n for terminal type -- 6 for 6053's). You can DISPLAY it, READ it back onto your screen, and if you modify it, WRITE it back out to screen #0. You can have several screens in one screen file; the limit depends on your available memory. There is a quick summary of the commands for SM (which you can only use with the ">>" prompt command line) in Table 6-2.

Table 6-2. Screen Maintenance Commands

Command	Action
READ *[screen]*	Reads the screen file and displays the specified screen number in *screen*. If *screen* not specified, it reads in the screen last referenced.
WRITE *[screen] [system,project,program]*	Reads the screen from CRT and outputs it to the screen file at the specified screen number in *screen*. If *screen* not specified, it outputs to last screen referenced. You can optionally include *system* name, *project* name, and *program* name for printouts of screens.
DISPLAY *[screen]*	Displays the specified screen number in *screen* without the format characters. If no *screen* is specified, it displays the last screen referenced.
PRINT *[filename]*	Prints the current screen to *filename* including *system, project* and *program)*. If no *filename* is specified, it prints the current screen to the line printer.
STOP	Stop an SM session.

Licensed Material-Property of Data General Corporation

Using Your Screen

You can use your screen in two ways: for File Maintanance, and for programs you write using UNFORM.SL, FORM.SL, VAL.SL, and PROTFORM.SL subroutines. (Before using FORM.SL, UNFORM.SL, PROTFORM.SL, and VAL.SL, read the descriptions in Chapter 7.) File Maintenance allows you to override a formatted screen with one of your own. You, however, would want to write your own programs to handle real-time data entry and retrieval. We show you suggested programming techniques for setting up such programs. We show a sample program that uses the screen in Figure 6-3 to input inventory data and uses dynamic record and indexing routines to store the data in a data file with an index.

Your programs should use the UNFORM.SL routines to flash error messages, position the cursor to unprotected fields for input, etc. UNFORM.SL and FORM.SL both have routines to handle data input: UNFORM.SL is for unformatted fields, and FORM.SL handles formatted fields and flags any errors. Both routines also output messages and data, and output spaces to clear fields, etc.

FORM.SL and PROTFORM.SL use UNFORM.SL and VAL.SL, so you'll probably need to ENTER all of them into your program. First you must DIMension the variables they need:

```
DIM X$(80),LINE$(132),SCRN$(200)
```

X$ is used for input of data and output of data from fields you must DIMension to 80 bytes. LINE$ is a scratch variable used to input and output lines from and to the screen file, and you must DIMension it to 132 bytes. SCRN$ is another scratch variable that describes each field set by SM, and it must be DIMensioned to at least 48 plus three times the number of fields on the screen. Other input variables to UNFORM.SL (and FORM.SL, PROTFORM.SL) are: F for the number of the field in the row, R for the row number (of F=row number *100 plus field number, and you don't need R), and E for the error code.

Always OPEN FILE your screen file on channel 15 (the subroutines expect to find it there). For example:

```
OPEN FILE (15,5),"SCREEN.S6"
```

SCREEN S.6 is open for input/output (random) on channel 15. When you're ready to set the keyboard (lock the keyboard, set the necessary STMA's, etc.) GOSUB to 9150 (UNFORM.SL) to set the keyboard if you're allowing lowercase, or GOSUB 9155 if you do not use lowercase. Now you can follow the sequence in Example 6-1.

In example 6-1, line 190 displays the screen on the operator's terminal. Line 200 sets the field and row, line 210 sets the output, and line 220 outputs a 0 to the unprotected field in the first column to tell the operator to type a numeric. Lines 260 and 290 output blanks to show the unprotected fields you want to be filled in.

Now we're ready to receive input from the first unprotected field. Example 6-2 reads field 1 in row 5.

```
.
.
.
0160 OPEN FILE[15],"SCREEN.S6"     :Open screen file.
0170 GOSUB 9150                    :Set keyboard, STMA's etc.
0180 LET SCRN=0                :Screen number 0.
0190 GOSUB 9700                :Display screen in SCRN.
0200 LET F=501                 :Row 5, field 1 (unprotected field).
0210 LET X=0                   :Initialize X.
0220 GOSUB 9500 :  A WRITEFORM     :FORM.SL, write X to field
0230 LET F=601      :Row 6, field 1.
0240 LET X$=""      :Null string for output (to output spaces).
0250 LET X$[1,30]=FILL$(32)  :Fill string with spaces.
0260 GOSUB 9550                :Output string to field F.
0270 LET F=701                 :Row 7, field 1.
0280 LET X$=" "                :Space in string for output.
0290 GOSUB 9550                :Output string to field F (UNFORM.SL).
.
.
.
```


Licensed Material-Property of Data General Corporation

Example 6-2

```
.
.
.
0300 LET F=501                  :Row 5, field 1.
0310 GOSUB 9650                 :Position cursor to field F.
0320 GOSUB 9300 :  Δ READFORM       :Read formatted input.
0330 IF LEN(X$)=4 THEN GOTO 0370  :Check length of input.
0340 LET X$="KEY FIELD OF INSUFFICIENT LENGTH -- MUST BE 4 LONG"
0350 LET E=0                        :Set error code to 0.
0360 GOTO 0410                      :410 outputs an error message.
0370 LET KEY$=X$                    :Assign KEY$., for index.
0380 GOSUB 7600 :  Δ VAL            :Convert X$ to number.
0390 IF E=0 THEN GOTO 0460          :If no error, go to 460.
0400 LET E=128                      :If error, make it error 128.
0410 GOSUB 9850                 :Output error message at bottom of screen.
0420 GOSUB 9750                 :Read a function key hit by operator.
0430 IF FUNC<>16 THEN GOTO 0410   :If not key you want,go back,try again.
0440 GOSUB 9890                 :If key is F11(value 16),clear error.
0450 GOTO 0300                  :Go back and read field.
0460 LET F=601                  :Row 6, field 1.
0470 GOSUB 9650                 :Position to field F.
0480 GOSUB 9300 :  Δ READFORM       :Read field (FORM.SL).
0490 LET DS$=X$,FILL$(32)       :Assign input to DS$.
0500 LET F=701                 :Row 7, field 1.
0510 GOSUB 9300 :  Δ READFORM       :Read formatted input (FORM.SL).
0520 LET T$=X$                  :Assign input to T$.
.
.
.
```

Example 6-2 reads the field in row 5, then checks the input to see if it is of sufficient length. If so, we assign the input to KEY$. If not, we assign an error message to X$, set the error code to 0 (to tell UNFORM.SL that we want to send our own error message), and go to line 410 which GOSUBs to UNFORM.SL's routine and sends an error message (or X$) to the bottom of the displayed screen. Line 420 is an UNFORM.SL routine that waits for a function key input. If FUNC is not 16, we go back and wait again for the right function key.

NOTE: If FUNC is returned with a value of 1 to 8, this value corresponds to function keys F1 to F8. If FUNC returns with a value of 14 to 16, these values correspond to function keys F9 to F11. If the key received is a value of 16 in FUNC (F11), then line 400 clears the error message, and line 450 returns us to the input routine, so that the operator can retype the screen.

After assigning the input to KEY$ on line 370, we GOSUB to VAL.SL to convert X$ to a number. If there is an error, then the operator must have keyed some nonnumeric character. At line 390, we test for the error, and if there is one, we assign error code 128 -- INVALID OR OUT OF RANGE FIELD to E, output that error message, and wait for the right function key which clears the error. The operator can now reposition to the field and try the input again.

Licensed Material-Property of Data General Corporation

Sample Program

The program in Example 6-3 will work if:

1. You set up your screen as shown in Figure 6-3.

2. You set up the PARAM records for the data and index files by using INITFILE. Make you data record "ITEM", physical or subfile, with a record size of 100 bytes per record and about 100 records. Make your index file "ITEM.IX", a physical or subfile (but not in the same physical file as ITEM), with a maximum of 6 records (sectors -- calculated by INDEXCALC) and 4 bytes per key. If these records are in PARAM, you'll have no trouble.

3. Finally, you must key in this program and then ENTER these subroutines: FORM.SL, UNFORM.SL, FINDFILE.SL, POSFL.SL, GETREC.SL, and VAL.SL.

The program in Example 6-3 solicits keyboard input to the screen, writes the data to ITEM, and adds the key to ITEM.IX. When you stop inputting, the program searches the index for the keys in proper ascending order and prints the data on the line printer.

The program in Example 6-3 checks errors either by checking the data input (line 330) or the value of E (line 390). The program sends its own error message by letting E = 0 (line 350), putting the error message in X$ (line 340) which is used by the subroutines FORM.SL, UNFORM.SL (and PROTFORM.SL, not used in this program), and going to the UNFORM.SL subroutine entry point 9850 (line 410). When you output an error, you usually want to wait until the operator hits a certain key before you clear the error (and sometimes positions back to the field to correct the error). FUNC returns a function key value (using UNFORM.SL's entry point 9750) in the range 1 to 8 for function keys F1 to F8, and in the range 14 to 16 for function keys F9 to F11. FUNC returns a -1 if a something other than a function key was pressed. We check for a FUNC value at lines 430, 700, and 710.

Licensed Material-Property of Data General Corporation

Example 6-3

```
0010 STMA 6,5                          :Do not acknowledge interrupts.
0020 DIM C1[1,3],X$[80],T9$[42],B$[544],D$[18]
0030 DIM KEY$[4],DS$[30],T$[1],LINE$[132],SCRN$[60]
0040 LET X$="ITEM"                     :Name of data file
0050 LET F%=0                          :ITEM will be logical file 0 in C1.
0060 GOSUB 7800                        :FINDFILE.SL to build C1.
0070 OPEN FILE[F%,0],X$                :Open data file.
0080 DIM REC$[C1[F%,3]]                :Record size from C1.
0090 LET C1[F%,0]=F%                   :Assign 0 column of C1.
0100 LET X$="ITEM.IX"                  :Name of index file.
0110 LET F%=1                          :ITEM.IX will be logical file 1 in C1.
0120 GOSUB 7800                        :FINDFILE.SL.
0130 OPEN FILE[F%,0],X$                :Open index file.
0140 LET C1[F%,0]=F%                   :Assign 0 column of C1.,assign index string.
0150 LET D$=CHR$(C1[F%,0],2),CHR$(C1[F%,1],4),CHR$(0,2),X$,FILL$(0)
0160 OPEN FILE[15],"SCREEN.S6"         :Open screen file.
0170 GOSUB 9150                        :Set keyboard, STMA's etc.
0180 LET SCRN=0                        :Screen number 0.
0190 GOSUB 9700                        :Display screen in SCRN.
0200 LET F=501                         :Row 5, field 1 (unprotected field).
0210 LET X=0                           :Initialize X.
0220 GOSUB 9500 :  Δ WRITEFORM         :FORM.SL, write X to field
0230 LET F=601      :Row 6, field 1.
0240 LET X$=""      :Null string for output (to output spaces).
0250 LET X$[1,30]=FILL$(32)  :Fill string with spaces.
0260 GOSUB 9550                        :Output string to field F.
0270 LET F=701                         :Row 7, field 1.
0280 LET X$=" "                        :Space in string for output.
0290 GOSUB 9550                        :Output string to field F (UNFORM.SL).
0300 LET F=501                         :Row 5, field 1.
0310 GOSUB 9650                        :Position cursor to field F.
0320 GOSUB 9300 :  Δ READFORM          :Read formatted input.
0330 IF LEN(X$)=4 THEN GOTO 0370  :Check length of input.
0340 LET X$="KEY FIELD OF INSUFFICIENT LENGTH -- MUST BE 4 LONG"
0350 LET E=0                           :Set error code to 0.
0360 GOTO 0410                         :410 outputs an error message.
0370 LET KEY$=X$                       :Assign KEY$., for index.
0380 GOSUB 7600 :  Δ VAL               :Convert X$ to number.
0390 IF E=0 THEN GOTO 0460             :If no error, go to 460.
0400 LET E=128                         :If error, make it error 128.
0410 GOSUB 9850                   :Output error message at bottom of screen.
0420 GOSUB 9750                   :Read a function key hit by operator.
0430 IF FUNC<>16 THEN GOTO 0410   :If not key you want,go back,try again.
0440 GOSUB 9890                   :If key is F11(value 16),clear error.
0450 GOTO 0300                    :Go back and read field.
0460 LET F=601                    :Row 6, field 1.
0470 GOSUB 9650                   :Position to field F.
0480 GOSUB 9300 :  Δ READFORM          :Read field (FORM.SL).
0490 LET DS$=X$,FILL$(32)     :Assign input to DS$.
0500 LET F=701                :Row 7, field 1.
0510 GOSUB 9300 :  Δ READFORM          :Read formatted input (FORM.SL).
0520 LET T$=X$                :Assign input to T$.
0530 LET REC$[1,2]=CHR$(1,2) :Build record string.,this is status.
0540 LET REC$[3,6]=KEY$       :This is key,
0550 LET REC$[7,36]=DS$,FILL$(32) :this is description,
0560 LET REC$[37,37]=T$       :and this is type.
```


Licensed Material-Property of Data General Corporation

```
0570 LET REC$[38,100]=FILL$(32)    :Fill remaining with spaces.
0580 LET F%=0                      :Logical file 0 is data file.
0590 LOCK 1,"ITEM",0,C1[F%,3]      :Lock record 0 of data file.
0600 GOSUB 8400 :  Δ GETREC        :Get available record.
0610 UNLOCK 1                      :Unlock record 0, GETREC.SL is through.
0620 GOSUB 9610 :  Δ POSFL         :Position to record R1 returned by GETREC.
0630 WRITE FILE[C%],REC$           :Write new record, POSFL returns C%.
0640 KADD D$,B$,KEY$,R1            :Add key to ITEM.IX with R1 pointer.
0650 IF R1<=0 THEN GOTO 0720       :If error, go to error routine.
0660 LET X$="F10=STOP   F8=CONTINUE"   :Operator chooses.
0670 LET E=0                       :Let error code = 0.
0680 GOSUB 9850                    :Output choice message X$.
0690 GOSUB 9750                    :Read a function key.
0700 IF FUNC=15 THEN GOTO 0760     :Looking for key F10 (value 15).
0710 IF FUNC=8 THEN GOTO 0200      :If F8, continue input.
0715 GOTO 0660                     :If neither, go back, try again.
0720 LET X$="<7>ERROR IN ADD<7>"   :Error message.
0730 LET E=0                       :Error code = 0.
0740 GOSUB 9850                    :Output error message from X$.
0750 STOP
0760 GOSUB 9175                    :Reset terminal for normal use.
0770 CLOSE FILE[15]                :Close screen file.
0780 OPEN FILE[2,1],"$LPT"         :Open line printer.
0790 LET KEY$=""                   :Null key.
0800 KFIND D$,B$,KEY$,R1           :Find null key (first key).
0810 LET R1=ABS(R1)                :Expecting a negative return.
0820 GOTO 0850
0830 KNEXT D$,B$,KEY$,R1           :Get next key.
0840 IF R1=0 THEN GOTO 0910        :If last key, go to end.
0850 LET F%=0                      :Logical file for data file.
0860 LET V%=2                      :2 byte offset into record.
0870 GOSUB 9612 :  Δ POSFL WITH OFFSET V%   :Position to record.
0880 READ FILE[C%],REC$[3,100]     :Read record associated with key.
0890 PRINT FILE[2],USING "T5,A4,2X,A30,2X,A1",REC$[3,6],REC$[7,36],REC$[37,37]
0900 GOTO 0830                     :Get next key.
0910 STMA 7,5                      :Reset no-IKEY to acknowledge interrupts.
0920 CLOSE FILE[0]                 :Close files.
0930 CLOSE FILE[1]
0940 CLOSE FILE[2]
0950 END
```

Sample Output Routine

Figure 6-4 shows a screen with unprotected fields used for output. Your program might position the cursor to the vendor number field at the top for the terminal operator to key in that value which is used to find the data record with the check information. Your program would follow a routine such as the one outlined in Figure 6-5 to output the data.

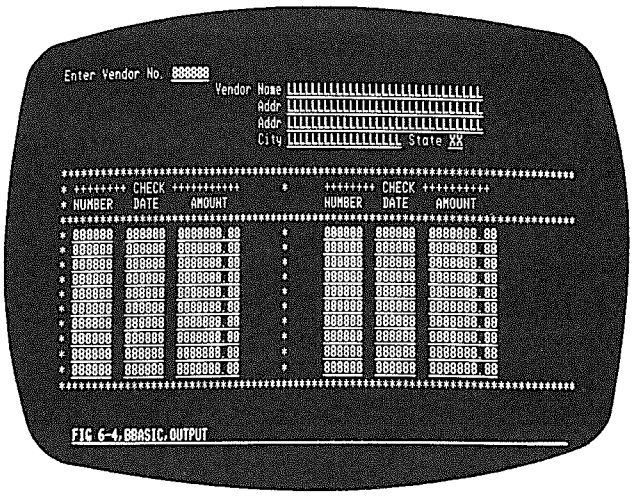


Figure 6-4. Screen for Sample Output Routine

Licensed Material-Property of Data General Corporation

1. DIM X$(80),LINE$(132),SCRN$(246)

   These are used by UNFORM.SL, FORM.SL, and PROTFORM.SL. SCRN$ is dimensioned to 246 for 66 fields used (48+(3*66)).

2. Dimension string for record, descriptor string for index, C1 array for logical files, etc.

3. LET X=0 initialize X (used by FORM.SL and PROTFORM.SL)
   LET E=0 Error variable, initialize (used by FORM.SL, UNFORM.SL and PROTFORM.SL).

4. Open data and index files, set up C1 array and descriptor string.

5. Open screen file (ours is SCREEN.S6) on channel 15 for random input/output.

6. LET SCRN=1 Let SCRN equal the screen number; ours is 1 in SCREEN.S6.

7. LET F=101 or LET R=1, LET F=1 For field 1 in row 1.
   LET X$=" " Spaces to be output to this field.
   GOSUB 9550 UNFORM.SL, to output spaces to this field.

8. LET F=101 for field 1 row 1.
   LET X$="" Make X$ null to receive input.
   GOSUB 9650 Position to field.
   GOSUB 9300 FORM.SL to read input.
   IF E=128 THEN ... If error, handle it by outputting the error message for code 128, or by sending out your own error message.

   a. GOSUB 9850 Send out error message # 128.
      GOSUB 9750 Wait for function key.
      IF FUNC<>16 THEN ... If not F(11), go back and try again.
      GOSUB 9890 If F(11), clear error message from screen.
      LET F=101 For field 1 in row 1.
      GOSUB 9650 Position to field.
      GOSUB 9300 Read field input again, and go through error routine again.

   b. LET E=0,X$="your message" To send out your own error message.
      GOSUB 9850 Send out error message in X$, then do function key routine above.

Figure 6-5. Sample Output Routine

Licensed Material-Property of Data General Corporation

9. Using the input value for vendor number, KFIND the key with that value, READ FILE the data record for that key, and assign "Name" output to X$.

```
LET R=201                    Use FORM.SL or UNFORM.SL to output X$ for "Name".
GOSUB 9500 or 9550
```

Get "Addr", "City", and "State" data from record. For each one:

LET F=field, R=row (or F=100*row number + field number).

```
GOSUB 9500 or 9550     To output X$ (data).
```

10. To output the two "CHECK" columns in one pass, use these FOR/NEXT nested loops:

```
FOR X=1100 TO 2000 STEP 100   :For each row,
   FOR I=1 TO 2               :Output 2 groups per line.

      LET F=X+(I-1)*3+1
      LET X=check number      :Set F to first field in group.
      GOSUB 9500              :Output X using FORM.SL
      check E for error
      LET F=F+1               :For next field.
      LET X=date of check
      GOSUB 9500              :Output X using FORM.SL
      check E for error
      LET F=F+1               :For next field.
      LET X=amount of check
      GOSUB 9500              :Output X using FORM.SL
      check E for error
   NEXT I                     :For next group.
NEXT X                        :For next row.
```

Figure 6-5. Sample Output Routine

Licensed Material-Property of Data General Corporation

Screen File Layout

SM builds and maintains a screen file for you to use with FORM.SL, UNFORM.SL, PROTFORM.SL, FM, or Screen Maintenance itself. If you follow the routines, you will never have to look at the screen file's layout. But if you want to do special things, you should be familiar with the layout as shown in Table 6-3.

In Table 6-3, section A is read into SCRN$ when you input/output to screen files. There are 24 line pointers, and each pointer is followed by the field definitions (XCOL, XWID, and XFLGS, all used by UNFORM.SL) for each line. Section B's record count tells how many screen literals (LINE$, used by UNFORM.SL) follow. Screen literals (LINE$) are data-sensitive lines that FORM.SL and UNFORM.SL read using INPUT USING. You only need to reference section C when you want screen label verification.

Table 6-3. Screen File Layout

Record Format

Section	Contents	Location	Size	Type
A. Field definitions section (SCRN$)		1	variable	string
1.	Line pointers (24; 1 per line)	1	48 (24*2)	integer
2.	Field definitions (1 per field)	25	# of fields*3	
	Field column (XCOL)		1	binary
	Field width (XWID)		1	binary
	Field flags (XFLGS)		1	binary
B. Screen literals section		1601	variable	
1.	Screen literal record count	1601	4	double
2.	Screen literal record (LINE$)		variable	string
C. Screen label information section		4873	128	
1.	Status (value = 1)	4873	2	single(%)
2.	Last modification time (hhmmss)	4875	4	double
3.	Last modification date (mmddyy)	4879	4	double
4.	Last modifier's account number	4883	6	string
5.	Project identification	4889	30	string
6.	System identification	4919	30	string
7.	Program identification	4949	30	string
D. Unused		4979-5000		

Record Size: 5000 bytes per record (1 record per screen)
First Record #: 0

End of Chapter

Licensed Material-Property of Data General Corporation

Chapter 7
Subroutines and Utilities

The keywords in this chapter describe either subroutines or utilities. Subroutines have a .SL extension added to their filenames. Utilities are actually program files (i.e., SAVEd), whereas subroutines are listing files (i.e., LISTed).

Whenever you want to use a subroutine, you must ENTER the subroutine in order to merge it with your program. Either wait until runtime to do this, or ENTER the subroutine and save the whole program. You then have to use GOSUB and the entry point(s) of the subroutine. You can LOAD, RUN, CHAIN, or SWAP to utilities or execute them from the BASIC CLI, except where noted.

The following keywords are described in this chapter:

ATTACH	Attach your terminal to a de-attach job.
CLI	RUN or SWAP to the BASIC CLI.
DELREC.SL	Delete a record in a linked-available-record file.
DIV.SL	Calculate extended percentage.
DOC	Print a series of text input files.
DOCTOC	Print a table of contents (used with DOC).
EDIT	Create and/or edit a text file.
FILESORT	Sort a data file.
FINDFILE.SL	Find a subfile and build C1 array.
FORM.SL	Formatted screen input/output.
GETCM.SL	Read the CLI command string in the common area (write your own CLI command).
GETREC.SL	Get next available record in a linked-available-record file.
IBUILD	Build an index from a sorted tag or data file.
INDEXBLD	Build an index file.
INDEXCALC	Calculate index file information.
INDEXPRT	Print an index file.
INITFILE	Initialize an index or data file.
INITINDEX.SL	Initialize an index file.
LOCKS	Display your current locks.
LSPEED	Set your terminal's speed.
MUL.SL	Calculate percent of a number.
OPEN	Open physical files (for subfiles).
PORTS	Display jobs on the system.
POSFL.SL	Position a file to a record.
PROTFORM.SL	Protect/unprotect a formatted screen field.
QFILESORT	Fast sort of a data file (ascending only).
RELINK	Clean up a linked-avaliable-record file.
RNAM	Rename program variables.
SCHANS	Display the system channel assignments.
TBUILD	Build a tag file.
TERM	Set a certain terminal keys.
UCHANS	Display your channel assignments.
UNFORM.SL	Unformatted screen input/output
VAL.SL	Convert a string of digits to a number.
VAR	list the variables in a program.
XBUILD	Build an index from a data file.

Licensed Material-Property of Data General Corporation

• *Subroutine*
Utility

ATTACH

Attach your terminal to a de-attached job.

Why Use It?

Use ATTACH to connect your terminal to a de-attached job. A de-attached job is a job running independent of any terminal. You can start a job executing at your terminal and then de-attach it from your terminal using the de-attach key set by STMA 4,0, the TERM program, or the default key CTRL-D (<4>). The system discards any terminal output from a de-attached job. If a de-attached job executes an INPUT statement, the de-attached job waits ten minutes for the INPUT, and if the INPUT is not answered, the de-attached job logs off. You'd use ATTACH to attach yourself to a job and supply INPUT data before the job logs off.

What It Does

ATTACH connects your terminal with the de-attached job specified when you answer the question JOB NUMBER:. If the de-attached job is waiting for input, the message -- JOB IS WAITING FOR INPUT -- appears as soon as ATTACH attaches to the job. If you try to ATTACH to a job that's not in use, a message appears telling you that the job is not in use. If you try to ATTACH to your own job, the message THIS IS YOUR JOB appears.

How To Use It

You can RUN, CHAIN, SWAP, or execute ATTACH from the CLI. ATTACH asks you for JOB NUMBER:; respond with the job number of the de-attached job you want to ATTACH to. You can find the job number by running the STAT utility program (see STAT in this chapter) or the PORTS utility program (see PORTS in this chapter).

ATTACH Example

```
*RUN "STAT" )

00     C  R  13  DOC     STAT      ABTLB6  03  0  3428     55.7  2507  14:47
01  I  D  R  13  OMI     SCRATCH   ABEAS3  01  0  8914     113.9 1462  15:49
02  I  C  R  13  DOC     MYPROG    ABTLB6  -1  0  3428     137.7 8807  14:53
```

The output from STAT tells us that we're job 00 -- running STAT -- and the de-attached job MYPROG is job 2. De-attached jobs will have a -1 in the column after the ID, and our ID is ABTLB6.

```
*RUN "ATTACH" )

JOB NUMBER: 2 )
JOB IS WAITING ON INPUT
? 123 )
.
.
.
```

We are now attached to MYPROG. MYPROG is waiting for an input, so in response to the ? prompt, we input 123.

Licensed Material-Property of Data General Corporation

Subroutine
- *Utility*

CLI

Execute the BASIC CLI.

Why Use It?

Use the BASIC CLI to perform many tasks not normally available in a BASIC language. All the BASIC CLI commands are listed in Chapter 8 in alphabetical order. You can execute a CLI command from keyboard mode by typing an exclamation point (!) and the command. This swaps to the CLI and passes the command through the common area, and when the command is finished executing it pops you back into keyboard mode. However, when you use the immediate (!) CLI execution, your common area is used and the response time is slow. If you want to preserve your common area, or if you want to execute more than one CLI command and get a faster response, use the CLI utility instead of the immediate (!) CLI execution.

What It Does

If you SWAP to CLI, then when you POP out of the CLI you'll return to the program that was in working storage before you SWAPped to CLI. If you RUN or CHAIN "CLI", your current program is not saved in working storage. The immediate (!) CLI execution swaps to CLI with a command string passed through common area, and if there is more than one command in the command string, response time is slower than if you explicitly SWAP, CHAIN, or RUN "CLI".

How To Use It

You can SWAP, CHAIN, or RUN "CLI". When you do, you are put into the BASIC CLI (with an exclamation point (!) prompt) and you can execute any BASIC CLI command. The BASIC CLI commands are listed in Chapter 8. You can use POP to get out of the BASIC CLI: if you SWAPped to CLI, POP puts you back in the program that issued the SWAP. If you CHAIN or RUN the CLI, POP puts you back in BASIC's keyboard mode.

CLI Example

```
*"CLI )

Swap to CLI.

!DISK )

                        USED          LEFT        CONTIGUOUS
SECTORS                 115,432       71,976      66,696
BYTES                   59,101,184    36,851,712  34,148,352

!GTOD )

17:34:12    5/15/78

!POP )
*

The program which was in working storage is still there.
```


Licensed Material-Property of Data General Corporation

DELREC.SL

Delete a record and put it on the deleted record chain.

Input Variables

F% logical file number of file in file characteristics array (C1).

C1 file characteristics array.

R1 record number of record you want to be deleted.

Output Variables

None.

Scratch Variables

X0% record status of record 0.

Y0% record status of first record on deleted record chain.

X0 new text available record.

Y0 next available record from record 0.

Z0 last record used from record 0.

Line Numbers

8600	only entry point to DELREC.SL
8600-8695	DELREC.SL occupies these line numbers.

Why Use It

Use DELREC.SL to delete a record in a dynamically allocated file, and put the record on the deleted record chain so that it can be reused.

What It Does

DELREC.SL flags record R1 in file F% (described in the C1 array) as deleted and adds it to the deleted record chain, where GETREC.SL can find it. DELREC.SL calls POSFL.SL to do this.

How To Use It

You must ENTER "DELREC.SL" and merge it with your program -- your program cannot use line numbers 8600 through 8695. At the point in your program where you want to execute DELREC.SL, you GOSUB to line 8600.

Example

```
0010 DIM C1(2,3)
.
.
.
0050 LET F%=1
0060 LET R1=5
0070 GOSUB 8600
.
.
.
```

In logical file number 1 (in C1 array), we delete record 5.

Licensed Material-Property of Data General Corporation

DIV.SL

Calculate extended percentage.

Input Variables

X devidend in the calculation of PCNT.

Y divisor in the calculation of PCNT.

Output Variables

PCNT percentage X is of Y where PCNT=(X*10000/Y).

Scratch Variables

I% loop counter.

Z1 temporary value of X.

Line numbers

7550 only entry point to DIV.SL.

7550-7590 DIV.SL occupies these line numbers.

Why Use It?

DIV.SL returns PCNT as the percent X is of Y. PCNT will be an integer, so you have to supply a decimal point if you want to print is as decimal; otherwise, you can print PCNT as an integer with a percent sign (%).

What It Does

Computes what percentage X is of Y. The calculation is iterative allowing X and Y to have as many as eight significant digits. If Y is input as zero, PCNT is returned with a value of zero -- no divide by zero error is generated.

How To Use It

You must ENTER "DIV.SL" and merge it with your program -- your program should not use line numbers 7550 to 7590. At the point in your program where you want to execute DIV.SL, you GOSUB to 7550 after supplying values for X and Y.

Example

```
*ENTER "DIV.SL" )
0010 LET X=25
0020 LET Y=100
0030 GOSUB 7550
0040 PRINT PCNT;"%"
*RUN )

25%
*
```


Licensed Material-Property of Data General Corporation

Subroutine
• *Utility*

DOC

Print a series of text input files.

Why Use It?

DOC is capable of producing printed output with automatic text justification, line centering, paging, and page heading. Documents produced by DOC are divided into chapters and sections, with an input file corresponding to each section.

What It Does

DOC outputs a text file either to the line printer, the terminal, or a file you specify. DOC also accepts keyboard input and outputs what you type. Text files must be created by the EDIT utility, and their filenames must take the form:

chapter number.section number (e.g., 1.12=chapter 1, section 12).

DOC searches for filenames with this form in ascending order from the first chapter and section specified to the last chapter and section specified. The text files must already have control commands in the text -- control commands do not print, they merely control the way the output will be printed. We summarize the control commands at the end of "How To Use It".

How To Use It

You can execute DOC by typing RUN "DOC" or by typing DOC from the BASIC CLI. DOC begins by asking:

FIRST CHAPTER: LAST CHAPTER:

If the first and last chapters you want printed are the same, DOC asks:

FIRST SECTION: LAST SECTION:

so you can print portions of a chapter. If your answers to FIRST CHAPTER and LAST CHAPTER are not equal, DOC prints the entire chapters in between, starting each chapter at section 0 (or the first section in the chapter). DOC then asks for:

OUTPUT FILES:

If you respond with a carriage return, the output file is assumed to be your terminal. You can specify any output file ($LPT for line printer).

DOC is now ready to receive input. First it reads default header information from a file DOC.HD (you can add information to DOC.HD using EDIT). Next, input is solicited from your keyboard by the prompt LINE:, and you can type or control commands that will be output before DOC outputs the text file sections. You can type text or control commands to the LINE: prompt, and stop by typing a carriage return to the LINE: prompt. DOC then outputs the text files. As DOC searches for files corresponding to sections in a chapter, any file missing in sequence causes DOC to assume it is at the end of the chapter, reset the section count to zero, and increment the chapter count by one.

Control commands allow you to change the default margins, page length, number of printed lines per page, and headings; and allow you to start a new page, center lines, specify conditional paging, and turn justification on and off. All control commands begin with a backslash (\). Any line in the text file (or in your keyboard input to the LINE prompt) that does not start with a backslash (\) is assumed to be text and will be printed.

Justifying Text

Start justifying text with the \STX command. All text you place after the \STX command and before the \ETX command is justified to the right and left hand margins (either default margins or margins set by the \MAR command). \ETX turns the justification off. If DOC encounters a single blank line while justifying, the previous line of text is not justified to the right. Single blank lines are not printed when justification is on, so paragraphs are not separated by a blank line when printed (even if they are separated by one blank line in the text file). To separate output paragraphs by a blank line, separate them with two blank lines in the text file.

Centering Lines

Start centering lines (a line is centered between the two default or set margins) with the \QC command. You can optionally specify in a \QC command, the number of lines following \QC to be centered, or turn off centering using the \EQC command. Either the number of lines you specify following \QC are centered, or all the lines following \QC are centered until an \EQC command is encountered.

Licensed Material-Property of Data General Corporation

Setting Margins

The \MAR command sets the left and right page margins. The format is:

\MAR left,right

For example, \MAR 10,50 sets the left margin to column position 10 and the right margin to column 50. The default left and right margins are 1 and 70.

Unconditional Paging

If you use the \PAGE command without the optional arguments, it immediately and unconditionally starts a new output page (page-up). With the arguments, the \PAGE command sets the page length, and you use this format:

\PAGE *[printed lines per page, lines per page]*

For example, \PAGE 50,60 specifies that the output page is 60 lines long and that only 50 lines will be printed on a page. The default values are 66 lines per page with 55 printed lines per page.

Conditional Paging

The \CPAGE command starts a new page at this point unless there are a specified number of lines after the \CPAGE command. This is useful when you're printing a table or chart and you either want the whole chart at the bottom of the current page or at the top of the next page, but you don't want the chart to be split up. The format for the \CPAGE command is:

\CPAGE *[number of lines].*

For example, \CPAGE 10 says to DOC, if ten lines will fit from here to the bottom of the page, do not page-up; if ten lines won't fit, page-up now. If \CPAGE does not contain a *number of lines* argument, DOC scans the following lines for an \ECPAGE command. If the number of lines between \CPAGE and \ECPAGE will fit at the bottom of the page (control command lines not counted), DOC will not page-up; if the number of lines won't fit, DOC will page-up at the \CPAGE command line.

Overprinting

The \NLF command combines two lines to make one output line (no line feed). The line of text following the \NLF control command is retained and the next line of text is merged with it. Characters of the first line replace blanks of the second line. For example:

```
\MAR 1,20
\NLF
\QC 1
COLUMN A
\MAR 21,40
\QC 1
COLUMN B
```

becomes

COLUMN A COLUMN B

Headings

DOC prints two lines of heading at the top of each page of output. A section heading or page heading optionally follows the first heading on the top line. The \HEAD command changes the default headings and column position of the section/page heading. The \HEAD command has the format:

\HEAD column number, heading text

where column number is the starting column of the section/page heading. The \HEAD command replaces the first heading with the second heading and replaces the second heading with heading text. If you include the string !D! in heading text, DOC replaces !D! with the current date in the form -- mm/dd/yy.

If your column number is zero, the section/page heading is suppressed. If your column number is positive, a section heading of the form SECTION *section* is printed, where *section* is a string of numbers separated by periods indicating the current chapter, section, subsection, etc. *Section* is set when a string of numbers (e.g., 12.22.1) is encountered in line 1 of the text and the first two numbers are equal to the current chapter and section. A negative column number prints a page heading following the first heading. The form of the page heading is PAGE chapter-page number, and if the current chapter is zero the form is PAGE page number.

Licensed Material-Property of Data General Corporation

Setting Page Numbers

The \SPAGE command sets the current page number. The format is:

\SPAGE number

where number is the current page's number. Zero is the default beginning page number.

Control Command Summary

Command	Function
\CPAGE *[n]*	Page-up to next page if *n* lines will not fit on this page. If you do not specify *n*, read the input file until an \ECPAGE is found, set *n* to the number of noncontrol lines read.
\ECPAGE	See \CPAGE.
\EQC	Turns off centering, see \QC.
\ETX	Turns off justification, see \STX.
\HEAD n, text	Replace first heading with second heading; replace second heading with text. Starting column of section/page heading set to the absolute value of n. Print a section head if n is positive, print a page head if n is negative, suppress section/page heading if n is zero.
\MAR 1,r	Set the left margin to 1, the right margin to r.
\NLF	Print the next line of text without line feed.
\PAGE *[p, l]*	If you include no optional arguments, page-up to the next page. If you specify *p* and *l*, set the number of printed lines per page to *p* and the number of lines per page to *l*.
\SPAGE n	Set the current page number to n.
\STX	Turn on justification, continue to justify until encountering an \ETX command.
\QC *[n]*	If an argument *n* is included, center the next *n* lines of text. If you do not specify *n*, center until encountering an \EQC.

Licensed Material-Property of Data General Corporation

DOC Example

The text file for section 1.1 that we created using the EDIT utility follows.

```
1.              WRITING AND RUNNING A BASIC PROGRAM

1.1             PREPARING A BASIC PROGRAM

1.1.1      INTRODUCTION

\STX
BASIC programs are made up of statements.  Each statement is preceded by an
integer that can be between 1 and 9999 inclusive.  The number you give to
a statement determines the order in which it is executed and listed.  For
example, two statements to be executed sequentially should be given
sequential (but not necessarily consecutive) numbers.


Each statement is on a separate line.  The programmer terminates each
line at the terminal with a carriage return.


Typing errors on the terminal can be corrected by using special control
keys (Note: these keys may be different on some terminals. They may also
have been modified by a program.)
\ETX

\MAR 10,60
\NLF
     1.
\STX
Pressing CONTROL and H at the same time
erases the last character typed
(RUBOUT on some terminals).  The cursor is
backed up on a CRT.
\ETX

\NLF
     2.
\STX
Pressing CONTROL and X at the same time deletes the entire
line.  A back slash (\) is printed, representing line
deletion; BASIC gives a carriage return/line feed.  The
programmer may then type a new statement.
\ETX
```


Licensed Material-Property of Data General Corporation

DOC (continued)

*RUN "DOC")

FIRST CHAPTER: 1) *LAST CHAPTER:* 1)
FIRST SECTION: 1) *LAST SECTION:* 1)
OUTPUT FILE: $LPT)
LINE:)

The following would appear at the line printer.

```
1.              WRITING AND RUNNING A BASIC PROGRAM

1.1             PREPARING A BASIC PROGRAM

1.1.1     INTRODUCTION

BASIC  programs are made up of statements.  Each statement is preceded
by  an  integer  that can be between 1 and 9999 inclusive.  The number
you  give  to a statement determines the order in which it is executed
and  listed.  For example, two statements to be executed sequentially
should  be given sequential (but not necessarily consecutive) numbers.

Each  statement is on a separate line.  The programmer terminates each
line at the terminal with a carriage return.

Typing  errors  on  the  terminal  can  be  corrected by using special
control  keys  (Note:  these  keys may be different on some terminals.
They may also have been modified by a program.)

     1.


          Pressing  CONTROL and H at the same time erases the
          last  character  typed  (RUBOUT on some terminals).
          The cursor is backed up on a CRT.

     2.


          Pressing  CONTROL  and  X  at the same time deletes
          the  entire  line.   A  back  slash  (\) is printed,
          representing  line deletion; BASIC gives a carriage
          return/line  feed.   The programmer may then type a
          new statement.
```


Licensed Material-Property of Data General Corporation

Subroutine
• *Utility*

DOCTOC

Print a table of contents.

Why Use It?

Use DOCTOC to print a table of contents for files used with DOC.

What It Does

DOCTOC searches the current directory for filenames of the form:

```
chapter.section
```

DOCTOC then searches each section for a line of text that has the form:

```
chapter.section text
```

where text starts at column 16. This line is then printed. Section zero is not indented; all other sections are indented by two spaces. A blank line separates each chapter. When a section is not found, the section count is reset to zero and the chapter count is incremented.

How To Use It

You can execute DOCTOC by typing RUN "DOCTOC" or from the BASIC CLI by typing DOCTOC. The program asks for FIRST CHAPTER and LAST CHAPTER, and then it asks for OUTPUT FILE. You can specify any output file, or answer with a carriage return if you want to have the output appear at your terminal. Your text file for each section should have a line of text in the form: chapter.section text where text starts at column 16 on a line. See DOC for an example of a text file in this form.

Subroutine
• *Utility*

EDIT

Create and/or edit a text file.

Why Use It?

You can create text, edit text files, and add comments to programs using EDIT. A frequent use of EDIT is to add comments to a listing file (source file) to maintain a program in character format without having the comments take up space in a SAVEd program (compile format). Any file created by EDIT can be printed using the BASIC CLI commands PRINT or BLDCOM/PRTCOM, or using the DOC utility program.

What It Does

When you execute EDIT, the editor comes up in command mode, indicated by a 000: prompt. This prompt is the current line, and it changes as text is keyed in to show the relative line you are working on. EDIT has three modes of operation: command, insert, and comment. Command mode allows you to open files to be edited, move or copy lines in the edit buffer, set parameters, search for a given character sequence, change characters to other characters, and input/output lines to and from the edit buffer. Insert mode allows you to key new text anywhere in your edit buffer. Comment mode is a quick and easy way you can add comments to source programs. You'll find a quick summary of all the commands at the end of "How To Use It".

You execute EDIT by typing:

```
RUN "EDIT" )
```

or

```
EDIT )
```

from the BASIC CLI. You are immediately put in command mode with the 000: prompt.

Command Mode

In order to conserve memory in the system, EDIT uses a disk resident buffer to contain the lines being operated on. This buffer is fixed in size to allow a maximum of 132 characters per line and 132 lines in the buffer. The commands that move data between the edit buffer and the file are: A, 0, OW, PAGE, END, and STOP. These will be explained in detail, but first we have to specify our input file.

Licensed Material-Property of Data General Corporation

EDIT (continued)

Specifying Input and Output Files

The first command you give is the command:

F filename

where filename is the name of your input file (if it exists) or the name of the file you are creating. Here's an example:

000: F 1.2

Here the filename is 1.2.

Using this command causes filename to be opened as the primary input file and

EDIT$$$process job

is opened as a temporary output file (process# is from SYS(25); job# from SYS(9)). When your editing session is completed and you use STOP or END, your temporary output file will be renamed to filename (primary input file's name).

Commands that reference files can have a colon (:) appended to the command causing the command to act on alternate input file. Thus, F: TEMP.SR opens the TEMP.SR as an alternate input file. A: brings in (append) lines from the alternate input file into the edit buffer, whereas A brings in lines from the primary input file. PAGE: outputs the buffer to the output file and brings in a new page from the alternate input file, whereas PAGE alone inputs from the primary input file. You can only have one temporary output file and that is renamed to the primary input file when you END or STOP.

File Input and Output

After specifying your input file(s), you must use the A command to bring lines of text into your edit buffer. The A command takes the form:

A [n]

where *n* is optional and is the number of lines you want brought into the buffer. If you don't specify *n,* A will bring in one page (160 lines, or the number you set by LINES). If you specify *n* lines, or if you already have lines in your buffer, you must be careful not to exceed the 132 maximum number of lines allowed in your buffer.

After editing, adding, deleting, etc. lines in your buffer, you output the buffer to the output file using the commands O, OW, or PAGE. O and OW are identical except that OW writes a page mark (consisting of the line -- :\P) as the last line that it outputs. The O command takes the form:

O *[n [,m]],*

where *n* and *m* are optional, and *n* is the number of lines from the current line to be output, or *n,m* where *n* through *m* are the exact lines to be output. If you don't specify *n* or *n,m,* O outputs the entire buffer to the output file. This also empties the buffer so that you can bring in more lines. Lines output from a buffer are automatically deleted from the buffer, so if you O *n* lines, or O *n,m* lines, those lines will be deleted when they are output, and the position of lines in your buffer will change.

The PAGE command combines the action of A and O commands. A PAGE command outputs the entire buffer to the output file and inputs one page (60 lines, or the number you set by LINES) from the primary input file. A PAGE: command inputs the page from the alternate input file.

SKIP and COPY commands

The commands SKIP and COPY act on the input and output files, but they do not change what is in the buffer. SKIP reads the number of specified lines from the input file and discards them. If you don't specify the number of lines, a page (60 lines or the number you set by LINES) is read and discard. SKIP refers to the primary input file, and SKIP: refers to the alternate input file.

COPY reads the number of specified lines from the input file and writes them to the output file without using the buffer. If you don't specify the number of lines, a page (60 lines or set by LINES) is read and copied to the output file. COPY bypasses the buffer, so if you have any lines in the buffer, they won't be changed, but if you output the buffer after copying lines to the output file, the lines output from the buffer will appear in the output file after the lines copied. (This is one way you can move lines in the file -- input lines to be moved ahead into the buffer, copy the lines that are supposed to be before the lines in the buffer, and then output the buffer.)

Licensed Material-Property of Data General Corporation

SKIP and COPY can optionally act on lines until a string is found. For example, if you specify SKIP until this is found, SKIP will discard lines until it finds the string until this is found, at which point it backs up the file pointer to save the line that has the string until this is found. You can then bring the line and the lines following it into the edit buffer. Your string does not have to be in quotes. Your string can also contain the dash (-) and asterisk (*) conventions used to match a string. The dash will match any number of characters occurring at the location of the dash; the asterisk will match any single character occurring at its location. For example:

This	will match this	but not this
-.*	ABCD.E A.B	ABCD.EF ABC
AB-CD*	ABCDE ABBBCDE	ABECDEF ABBBCDEF

Setting Tabs

The SET command sets up to eight arbitrary tab stops for you, and you can tab to each tab stop using the TAB key. For example:

```
SET 5,10,20,35,42,50
```

sets tab stops at locations 5, 10, 20, 35, 42, and 50. You can then use the TAB key to get to these locations when you're in insert mode or comment mode.

Lowercase and Control Characters

The command ALC allows lowercase characters to be inserted, and the command ACC allows control characters to be inserted. By default, you are not allowed to type lowercase and control characters even if they already exist in the input file. ALC and ACC allow yu to insert these characters into your edit buffer. The DLC and DCC commands restore the default condition; i.e., they disallow lowercase and control characters.

Locating strings

Use the L and LC commands to locate strings in the edit buffer. The LC command will find all occurrences of the specified string and display them at your terminal (if you're not in brief mode, discussed later). The search begins with the first nonspace character in the string and continues up to the carriage-return delimiter of the string. For example:

```
LC GOSUB 9610
```

will find all occurrences of the string: GOSUB 9610. The L command will find only the first occurrence of the string.

To find all occurrences of strings that match a search string using dash (-) and asterisk (*) conventions, use the FIND command. The asterisk will match any single character occurring in the position of the asterisk, and the dash will match any number of characters occurring in the position of the dash (for examples, see the section "SKIP and COPY Commands". For example:

```
FIND -ABC-
```

will find all occurrences of any line that has the characters ABC together in a sequence (e.g., LET X$="RSJKL MN OPABCDR XY" in such a line).

```
FIND *ABC*
```

will find all occurrences of any line that has a five character string with ABC as the second, third, and fourth characters (e.g., LET X$="RSJKL RABCK OPAFG" is such a line).

Current Lines

Most commands operate on the current line of the edit buffer -- the line indicated by the prompt (e.g., the 003: prompt means the current line is line 3). You can change the current line by typing as a command to the prompt the desired line. For example, if you're at line 32 and you want to go to line 21, you'd type:

```
032: 21 )
021: THIS IS LINE 21.
021:
```

The line you go to is automatically displayed unless you're in brief mode (described later). If the line you want is higher than the last line in the buffer, you'll be put at the last line of the buffer and you'll get an error message.

Licensed Material-Property of Data General Corporation

EDIT (continued)

You can type the line you want, as we do above, or you can type a plus (+) or a minus sign (-) prefixed to a number, and the number is used as an offset. For example,

032: +5)
037: THIS IS LINE 37.
037: -10)
027: THIS IS LINE 27.
027:

Another way to change the current line is to use the T, B, and N commands. The T command moves you to the top of the buffer:

032: T)
000:

Line 0 of the buffer never contains an actual text line -- they start at line 1. You go to line 0 so that you can insert lines before existing line 1, or so that you can insert lines into an empty buffer.

The N command moves you to the next line after the current line:

000: N)
001: THIS IS THE FIRST LINE.
001:

The B command moves you to the bottom (last line) of the edit buffer and automatically puts you in insert mode so that you can add lines after the last line of your buffer:

Whenever you use the B command, or answer a carriage return to the last buffer line prompt, you are automatically put in insert mode (insert mode is described later).

001: B)
060: THIS IS THE LAST LINE OF THE CURRENT BUFFER.
1061: I AM INSERTING LINE 61.)
1062:

Licensed Material-Property of Data General Corporation

Changing Strings

You can change the contents of the current line using the C command. It takes the form:

C /string$_1$ /string$_2$

where / is a string delimiter. You can use any printable character (except a space) for a delimiter if the character is not used in string$_1$ or string$_2$. In our next example we use * as a delimiter, but in all other examples we'll use / as the delimiter to minimize confusion. Here's our first example:

005: C *ABC*XYZ)

Our current line is line 5. This command changes the first occurrence of ABC in the current line (line 5) to XYZ. To change all occurrences of string$_1$ in the current line, append a G to the command like this:

C /string$_1$ /string$_2$ /G

Therefore,

005: C /ABC/XYZ/G)

changes all occurrences of ABC in the current line to XYZ. To change *all* occurrences of string$_1$ in the edit buffer to string$_2$, use an A instead of a G:

C /string$_1$ /string$_2$ /A

Therefore,

005: C /ABC/XYZ/A)

changes *all* occurences of ABC in the edit buffer to XYZ, no matter what the current line was. You can also specify a range of lines, or specify a line increment from the current line. To change the first occurrence of string$_1$ from lines n_1 to n_2, use this form:

C /string$_1$ /string$_2$ / n_1 n_2

Therefore,

005: C /ABC/XYZ/25 35)

changes the first occurrences in each line of ABC to XYZ for lines 25 through 35, regardless of the current line. To change the first occurrence of string$_1$ for a number of lines (plus or minus) from the current line, use a line increment argument *i:*

C /string$_1$ /string$_2$ / *[-]* i

Therefore,

005: C /ABC/XYZ/ 10)

changes the first occurrence in each line of ABC to XYZ for the next 10 lines from the current line (including the current line), which means from line 5 through line 14.

To change *all* occurrences of string$_1$ to string$_2$ for each line between and including n_1 and n_2, or for *i* number of lines from the current line, use a G with the form, like this:

C /string$_1$ /string$_2$ /G $\left\{ \begin{matrix} [-]i \\ n_1\ n_2 \end{matrix} \right\}$

Therefore,

005: C /ABC/XYZ/G -3)

changes *all* occurrences of ABC to XYZ from line 3 through line 5, and

005: C /ABC/XYZ/G 25 35)

changes *all* occurrences of ABC to XYZ from line 25 through line 35, regardless of the current line.

Moving and Copying Lines

You can move or copy lines from one location in the edit buffer to another location in the edit buffer. The MOVE command moves line$_1$ through line$_2$ to the line following the current line, inserting them and numbering them accordingly. The original lines are deleted and the space they occupied is closed up -- lines following the original lines are renumbered to take up the space. For example,

010: MOVE 4,7)

moves lines 4 through 7 to lines 11 through 14, then renumbers the lines to use up the space left from deleting lines 4 through 7 so line 10 becomes line 6, line 11 (formerly line 4) becomes line 7, line 12 (formerly line 5) becomes line 8, etc.

Licensed Material-Property of Data General Corporation

EDIT (continued)

The DITTO command moves lines but does not delete the original lines. The DITTO command looks like the MOVE command, but

010: DITTO 4,7)

copies lines 4 through 7 to lines 11 through 14, moving original line 11 to line 15, etc., but keeps the original lines 4 through 7 untouched.

Replacing Lines

You can replace the current line of text with a new line of text using the R command. For example,

035: R THIS IS NEW LINE 35.)

replaces the current line's text (line 35) with THIS IS NEW LINE 35. To change any line of text in the edit buffer, use the line number:

001: 35:THIS IS NEW LINE 35.)

This changes the text of line 35 (regardless of the current line) to THIS IS NEW LINE 35.

Deleting Lines

The D command deletes the current line. Thus,

035: D)

deletes line 35 from the edit buffer. The D command followed by one number uses the number as an increment from the current line. Thus,

035: D 10)

deletes line 35 through line 44 (+10 lines from 35). Another example,

035: D -5)

deletes line 31 through 35 (-5 lines from 35). The D command followed by two numbers uses the two numbers as edit buffer lines to be deleted. Thus,

035: D 10,20)

deletes lines 10 through line 20 regardless of the current line.

Inserting a Line

You can insert one line following the current line with the I command (do not confuse with *insert mode,* discussed next), and you can insert one line *before* the current line with the IB command. For example,

035: I INSERTS THIS LINE AFTER LINE 35 (LINE 36).)

The IB command, however, inserts the line *before* the current line. For example,

035: IB INSERTS THIS LINE BEFORE LINE 35 MAKING THIS LINE 35.)

Licensed Material-Property of Data General Corporation

inserts before current line 35 making the new line 35 and making the current line (35) line 36.

To insert a line anywhere in your edit buffer, specify the line number as a command with a plus sign (+) to insert the line *after* the specified line, or a minus sign (-) to insert the line *before* the specified line. Thus,

```
035: +27 THIS IS NEW LINE 28. )
```

inserts THIS IS NEW LINE 28. after line 27 regardless of the current line, and

```
035: -26 THIS IS NEW LINE 26. )
```

inserts THIS IS NEW LINE 26. before line 26, making it the new line 26 (and renumbering old line 26 to line 27, etc.). To insert more than one line, use insert mode.

Insert Mode

You can type new lines into the edit buffer using insert mode. To enter insert mode, use your interrupt key (IKEY), usually the ESCape key (or a key set by the TERM utility or STMA 4,6). A new prompt consisting of the current line with an I prefix will be displayed. For an example, see Figure 7-1.

Insert mode always inserts lines after the current line (where you hit the interrupt key). Each new current line has the I prefix. To get out of insert mode, use the interrupt key again. Be careful not to insert more lines than the edit buffer will hold (the edit buffer holds 132 lines maximum).

Comment Mode

When you are commenting source (listing) programs, use comment mode. The COM command will put you in comment mode, and you use the interrupt key (IKEY) to get out of comment mode. Comment mode basically allows you only to append text to existing lines. When you enter comment mode, the current line is displayed with a C prefix and the cursor is put to the end of the line of text where you can append new text. All of our program examples in this manual using the colon (:) type comments use this feature.

Colon comments to a program are only stored in a character version of your program (not in the program file). That's why we usually use a .SR or .LS extension to denote character-format files vs. program files. Figure 7-2 shows an example using comment mode on file PROGRAM.LS.

```
035:  ikey
I036: THESE ARE NEW LINES WE ARE TYPING AFTER LINE 35. )
I037: THE LAST ONE WAS LINE 36, AND THIS IS LINE 37. )
I038: THIS IS LINE 38. USE IKEY TO GET OUT OF INSERT MODE. )
I039: ikey
039:
```

Figure 7-1. Using Insert Mode

```
000:  F PROGRAM.LS )
000:  A 20 )
020:  ALC )
020:  SET 30,40,50,60 )
020:  1 )
001:  COM )
C001: 0010 DIM C1(2,3) (use tab key)     :Dimension C1 to be a 3*4 array. )
C002: 0020 DIM X$(10)                    :Dimension X$ to max filename size. )
C003: 0030 LET X$="SUB1"                 :SUB1 is a subfile in MASTER, )
C004: 0040 LET F%=0                      :and is logical file 0. )
C005: 0050 GOSUB 7800                    :Go to FINDFILE.SL subroutine. )
C006: 0060 GOSUB 0200 ikey
006:
```

Figure 7-2. Using Comment Mode

Licensed Material-Property of Data General Corporation

EDIT (continued)

Note that we used the SET command to set tabs, and we used the Tab key to tab over to our predefined tab locations. We also allow lowercase characters to be typed by using the ALC command. Note than in comment mode, when you are commenting programs you must remember to type a colon (:) first to mark the comment. Otherwise, if we did an ENTER "PROGRAM.LS" to interpret our listing program, the system would think that our comments were actual BASIC statements, and we'd get errors. When you want to interpret a source or listing program (using ENTER to bring in the character-formatted program) to make it executable (i.e., binary format like a program file), the comments stay in the .LS file and do not take up working storage space, nor are they copied from .LS into the program file.

Brief and Verify Modes

Normally, commands that change the current line number or modify a line will display the new line automatically. You can suppress this action (verify mode) by setting brief mode with the BR command. Verify mode can be restored with the V command.

Stopping an Edit Session

When you're finished editing a file (or a buffer), use the STOP or END commands. END finishes writing the buffer and moves the remaining input file to the output file, giving the output file an edited version of the input file. The END command will then delete the input file and rename the output file to the name of the input file.

STOP executes an END, then exits from the EDIT utility. You don't need to execute an END yourself; STOP automatically does this.

OMIT stops the EDIT utility without changing the input or output file: use OMIT if you made a grave error in your buffer that you can fix, if you forgot to output a buffer, or if you made some other error you can't recover from. OMIT keeps both the input and the output file opened, and it will not rename the output file to the input file. The temporary output filenames are:

EDITprocessor#job#.TM

EDIT$$$processor#job#

where processor# comes from SYS(25) and job# comes from SYS(9). These files are not deleted, so you can examine them after an OMIT.

Table 7-1. Command Summary for EDIT

Command Format	Function
*[±]*line	Go to line line, or go forward (+) or backward (-) line number of lines.
*[±]*line text	Insert text after (+) line, or before (-) line.
*[±]*line:text	Replace line with text, or insert text after (+) line or before (-) line.
A *[:][n]*	Bring in (append) *n* lines or entire page to edit buffer from input file or from alternate (:) input file -- entire page is 60 lines or set by LINES.
ACC	Allow control characters.
ALC	Allow lowercase characters.
B	Go to the bottom of the edit buffer (last line) and select insert mode.
BR	Brief mode, opposite of V.
C /string1 /string2 *[/{A / G}]*	Change first occurrence of $string_1$ to $string_2$ in the current line, or change all occurrences of $string_1$ *(G)* in the current line, or change all occurrences of $string_1$ for all lines *(A)*.
C /string1 /string2/*[G]*[{n, n2 / ± i}]	Change first occurrence of string2 to string2 for lines n2 through n2, or for + or - i lines from the current line, or change all occurrences *(G)*.

Licensed Material-Property of Data General Corporation

Table 7-1. Command Summary for EDIT (continued)

Command Format	Function
COM	Select comment mode.
COPY *[:]*n	Copy n lines of input file or alternate (:) input file to output file without changing the buffer.
COPY *[:]*string	Copy lines of input file or alternate (:) input file to output file until string is found (* and - allowed), without changing the buffer.
D $\left[\left\{\begin{matrix}\pm n \\ line_1, line_2\end{matrix}\right\}\right]$	Delete the current line (no arguments), delete *n* number of lines following (+) or previous to (-) current line, or delete line *line*$_1$ through *line*$_2$.
DCC	Disallow control characters.
DITTO line$_1$, line$_2$	Copy lines line$_1$ through line$_2$ to the lines immediately following the current line (does not delete line$_1$ through line$_2$).
DLC	Disallow lowercase characters.
END	Output current buffer and copy remaining input file to output file, and rename output file to input file (deleting input file).
F *[:]*filename	Specify input file or alternate (:) input file.
FIND string	Find all remaining occurrences of string in the edit buffer (* and - allowed).
I string	Insert string in the edit buffer after the current line.
IB string	Insert string in the edit buffer before the current line.
L string	Locate the next occurrence of string in the edit buffer.
LC string	Locate all remaining occurrences of string in the edit buffer.
LINES n	Set the number of lines to n to be brought into the edit buffer (a page). Affects commands A and PAGE.
MAX n	Set the maximum line length to n (default is 132 characters).

Licensed Material-Property of Data General Corporation

Table 7-1. Command Summary for EDIT (continued)

Command Format	Function
MOVE $line_1$, $line_2$	Move lines $line_1$ through $line_1$ to the lines immediately following the current line, then delete $line_1$ through $line_2$.
N	Go to and display the next line in the edit buffer.
O $\left[\left\{\begin{matrix}[\pm]n \\ line_1, line_2\end{matrix}\right\}\right]$	Output *n* lines, or the following (+) or previous (-) *n* lines, or $line_1$ through $line_2$, or, if you specify no arguments, all lines in the edit buffer to the output file.
OMIT	Stop EDIT, leave temporary output files open, leave input files open and unchanged.
OW $\left[\left\{\begin{matrix}[\pm]n \\ line_1, line_2\end{matrix}\right\}\right]$	Same as O, but puts a page mark (:\P) on the last line.
P!	Print all lines in the edit buffer.
P $\left[\left\{\begin{matrix}[\pm]n \\ line_1, line_2\end{matrix}\right\}\right]$	Print n lines, or the previous (-) or following (+) n lines, or print $line_1$ through $line_2$
PAGE *[:]*	Output current buffer and bring in the next page (60 lines or a number set by LINES) from the input or alternate (:) input file.
R string	Replace the current line with string.
SET loc_1 *[,loc_2 ...loc_8]*	Set tab locations at column loc_1, etc.
SKIP *[:]*n	Skip n lines of input or alternate (:) input file and discard.
SKIP *[:]*string	Skip lines of input or alternate (:) input file until string is found (* and - allowed).
STOP	Execute an END and terminate the EDIT utility.
T	Go to the top (first line) of the edit buffer.
V	Verify mode, always on unless suppressed by BR (brief mode), to display lines acted on by editing commands.

Licensed Material-Property of Data General Corporation

Subroutine
- *Utility*

FILESORT

Sort a data file.

Why Use It?

FILESORT sorts a disk file in ascending or descending order based on key fields you define: you can have multiple key fields in each data file. FILESORT allows signed integer fields as keys, checks the status of deleted records, and supports your own compate routines; whereas QFILESORT only sorts in ascending order on a single unsigned key (see QFILESORT).

What It Does

FILESORT sorts a data file in place on the disk. FILESORT will sort on multiple keys that you specify, and tests the status of deleted records to sort them to the end of the file.

How To Use It

You have to SWAP to FILESORT from your program and pass arguments through the common area (BLOCK WRITE). You must open your data file and pass its channel number as an argument. All arguments are considered binary, and bit flags are numbered 0-7. You build a string with these arguments in these locations:

Substring Location	Size	Contents
1,1	1	Channel number of file you want to be stored.
2,5	4	Number of records you want to be sorted.
6,7	2	Record size in bytes.
8,11	4	Byte offset to first record if data file is a subfile.
12,12	1	Sort modifier=1 to check for deleted records.
13,13	1	Number of key fields in record.
14,18	5	Key field descriptors:
14,15	2	First byte of key (1 is first byte of record).
16,17	2	Last byte of key.
18,18	1	Key Type: bit 0=1 Sort field in descending order, else ascending. bit 1=1 Signed binary field, else unsigned. bits 2-7 Subroutine vector (array) for special compare routine(s).
19,-	5	next key field's descriptors.

With the string built, BLOCK WRITE the string to common and SWAP to FILESORT. BLOCK READ the returned arguments:

Substring Location	Size	Contents
1,2	2	Error code: 0=successful, a negative value is an RDOS/BASIC/IO error, and a positive value is a BASIC error, see Appendix C.

Test this substring to check for an error: the rest of the string is unused.

Examples

```
1000 LET X$=CHR$(0),CHR$(10000,4)
1010 LET X$(0)=CHR$(32,2),CHR$(0,4)
1020 LET X$(0)=CHR$(1),CHR$(1)
1030 LET X$(0)=CHR$(1,2),CHR$(4,2)
1040 LET X$(0)=CHR$(3),FILL$(0)
1050 BLOCK WRITE X$
1060 SWAP "FILESORT"
1070 BLOCK READ X$
1080 IF ASC(X$(1,2)) THEN GOTO 2000
.
.
.
```

Lines 1000 to 1040 fill the string to be sent to FILESORT. FILESORT returns arguments to the common area, and we check the first two bytes of the returned string on line 1080 to see if an error occurred.

Licensed Material-Property of Data General Corporation

FINDFILE.SL

Find a subfile and build C1 array.

Input Variables

X$ subfile name (logical name, no imbedded spaces).

F% logical file number for C1 array dimension.

Output Variables

X$ physical file's name in which the subfile resides.

C1 file characteristics array (only columns 1, 2, and 3 are set, column 0 is not set). See Chapter 4.

C% next available channel number where you can open a file.

Scratch Variables

T9$ subfile entry read from PARAM file, must be DIMensioned to at least 42 bytes.

Z pointer to subfile entries in PARAM file.

Line Numbers

7800 only entry point to FINDFILE.SL.

7800-7845 FINDFILE.SL occupies these line numbers.

Why Use It?

FINDFILE.SL builds your C1 array for you, and gives you the physical filename and next available channel number where you can open your subfile. You need a C1 array if you're using subfiles (see Chapter 4).

What It Does

FINDFILE.SL looks in the PARAM file for the subfile's entry, and uses the information to build columns 1, 2, and 3 for row F% (logical file number) of the C1 array, and it sends you the array, the physical file's filename, and the next available channel number. If the subfile entry doesn't exist or isn't found, FINDFILE.SL assumes you sent it the name of a physical file, and it asks you for the byte offset in the file for the first record, the record size, and the number of records in the file.

How To Use It

You must ENTER "FINDFILE.SL" and merge it with your program -- your program cannot use line numbers 7800 to 7845. You must DIMension your C1 array (see Chapter 4), DIMension T9$ to at least 42 bytes, DIMension and assign a subfile name to X$, assign a logical file number (row in C1 array) to F% and GOSUB to 7800. You can then use your C1 array, and X$ and C% in an OPEN FILE statement to open your subfile.

FINDFILE.SL Example

```
0020 DIM X$(20),T9$(42),C1(1,3)          :C1 for 2 logical files (0-1).
0040 LET X$="SUB1",FILL$(0)              :For subfile SUB1.
0050 LET F%=0                            :Logical file 0 in C1 array is SUB1.
0060 GOSUB 7800                          :Go to FINDFILE.SL to fill C1 array.
0070 OPEN FILE (C%,0),X$                 :Open file using C% from FINDFILE.SL.
0080 LET C1(F%,0)=C%                     :Fill 0 column of C1 array.
.
.
.
```


Licensed Material-Property of Data General Corporation

FORM.SL

Formatted screen input/output.

Input Variables

X numeric value to be output to a screen field. This subroutine uses UNFORM.SL and the same input arguments as you use for UNFORM.SL.

Output Variables

X numeric value input from a screen field.

E error code.

Scratch Variables

Same as UNFORM.SL.

Line Numbers

9300	entry point: inputs the screen field to X$ if a string field, or to X if a numeric field.
9500	entry point: outputs X$ (if string) or X (if numeric) to a screen field.
9100-9895	line numbers used by FORM.SL, UNFORM.SL, and PROTFORM.SL.
7600-7659	you need VAL.SL subroutine.

Why Use It?

Use FORM.SL instead of UNFORM.SL to do an error check on fields input and output -- to see if the data conforms to the specifications you set in the screen field (for your screen file -- see Chapter 6).

What It Does

FORM.SL uses routines in UNFORM.SL and VAL.SL to read input data from screen fields and to output data to screen fields. FORM.SL provides an error check for numeric fields. If the numeric value input from a screen field is in error, E is returned with a value or 128; if the conversion is successful, E is returned with a value of 0. String input is converted to uppercase and/or crammed. Numeric output to a screen field will conform to the field's specifications, and string fields will be uncrammed if necessary.

How To Use It

You must ENTER "FORM.SL" and "UNFORM.SL" and follow the requirements for UNFORM.SL. Use entry point 9300 to input a value from the screen field, and entry point 9500 to output a value to the screen field. You must initialize X if you're inputting and/or outputting numeric values. You must also position correctly to the screen field (see UNFORM.SL).

FORM.SL Example

```
      .
      .
      .
0190 GOSUB 9700                  :Display screen using UNFORM.SL.
0200 LET F=501                   :Row 5, field 1 (unprotected field).
0210 LET X=0                     :Initialize X.
0220 GOSUB 9500                  :FORM.SL, write X to field (as a prompt).
      .
      .
      .
0300 LET F=501                   :Row 5, field 1.
0310 GOSUB 9650                  :Using UNFORM.SL, position to field.
0320 GOSUB 9300                  :FORM.SL, read formatted input.
0325 IF E THEN GOTO 0410         :If error, go output error message.
      .
      .
      .
```


Licensed Material-Property of Data General Corporation

GETCM.SL

Read the CLI COM file.

Input Variables

Q9 byte pointer to beginning of field (in COM file) you want to be read.

T9$ string (DIMensioned to 512 bytes) to receive COM file produced by CLI in the common area.

Output Variables

Q9 byte pointer to beginning of next field.

X$ string containing field read.

S switches associated with the field read (-1 if end of command line).

Line Numbers

7550 entry point to initialize routine variables and read common area into T9$.

7500 entry point to read field pointed to by Q9. Returns a -1 in S if at end of command line.

7500-7560 GETCM.SL occupies these line numbers.

Why Use It?

When the BASIC CLI does not recognize a command, it builds a command file (COM file) in the BASIC common area and SWAPs to the filename in the first field of the COM file: RDOS/DOS CLI works the same way (see *RDOS/DOS CLI User's Manual,* Appendix D). GETCM.SL extracts fields from the COM file and returns the field and any switches associated with it. You can use GETCM.SL in a program that will generate your own CLI command.

What It Does

Entry point 7550 initializes routine variables and gets the COM file from the common area and puts it in T9$. Entry point 7500 reads the field in T9$ pointed to by Q9, returning in S the switch values:

Switch	Bit position	Value returned in S
/A	31	-2147483648
/B	30	1073741824
/C	29	536870912
/D	28	268435456
/E	27	134217728
/F	26	67108864
/G	25	33554432
/H	24	16777216
/I	23	8388608
/J	22	4194304
/K	21	2097152
/L	20	1048576
/M	19	524288
/N	18	262144
/O	17	131072
/P	16	65536
/Q	15	32768
/R	14	16384
/S	13	8192
/T	12	4096
/U	11	2048
/V	10	1024
/W	9	512
/X	8	256
/Y	7	128
/Z	6	64

For example, if switches for a field were /B/K/L, then the value returned by GETCM.SL for S would be 1076887552 (bits 30, 21 and 20).

How To Use It

To use GETCM.SL you must ENTER "GETCM.SL" into your program, use entry point 7550 to read the common area into T9$, and use entry point 7500 to read the field and extract the field and its switches.

Licensed Material-Property of Data General Corporation

GETREC.SL

Get next available record.

Input Variables

F% logical file number (row in C1 array for this file).

C1 file characteristics array (see Chapter 4).

Output Variables

R1 record number of next available record (-1 if file is full).

Scratch Variables

X0% record status of record 0.

Y0% record status of first record on the deleted record chain.

X0 new text available record.

Y0 next available record from record 0.

Z0 last record used from record 0.

Line Numbers

8400 only entry point for GETREC.SL.

8400-8495 GETREC.SL occupies these line numbers.

9610-9645 GETREC.SL uses POSFL.SL, POSFL.SL occupies these line numbers.

Why Use It?

Use GETREC.SL to find the next available record you can write to in a linked-available-record file (dynamic allocation). Linked-available-record files maintain their own space if you use GETREC.SL and DELREC.SL.

What It Does

GETREC.SL uses record 0 and the record status of each record in a linked-available-record file (dynamic allocation) to find the next available record you can write to -- either the next record in a deleted record chain or the lowest numbered unused record. GETREC.SL then uses POSFL.SL to position to the record.

How To Use It

You must ENTER "GETREC.SL" and "POSFL.SL" into your program. You must also build the C1 array (see Chapter 4), and assign the logical file number (row in the C1 array) for the file to F%. You GOSUB to 8400, and GETREC.SL positions you to record R1, so you can WRITE FILE to record R1.

GETREC.SL Example

```
0020 DIM X$(20),T9$(42),C1(1,3)          :C1 for 2 logical files (0-1).
0040 LET X$="SUB1",FILL$(0)
0050 LET F%=0
0060 GOSUB 7800                          :FINDFILE.SL to build C1 array.
0070 OPEN FILE (C%,0),X$
.
.
.
0100 LET F%=0                            :Logical file 0 (SUB1).
0110 GOSUB 8400                          :Go to GETREC.SL for available record.
0120 WRITE FILE (C%),REC$                :Write record.
.
.
.
```


Licensed Material-Property of Data General Corporation

Subroutine
- *Utility*

IBUILD

Build an index file from sorted tag or data file.

Why Use It?

IBUILD is faster than KADDs because it builds an index in one pass. Your input file must be a data file or tag file sorted by key in ascending order. INDEXBLD may use IBUILD. You can also specify a blocking factor to repack an index file. IBUILD is designed to be SWAPped to from your program.

What It Does

IBUILD reads the arguments you send into the common area. It then uses the sorted data or tag file to build in index using the blocking factor you specified. If your input file is a tag file, IBUILD assumes the key will start at the first byte of the tag file record, and the record pointer associated with the key will be the four bytes immediately following the key. If your input file is a sorted data file, you have to specify the starting and ending byte of the key, and IBUILD computes the data record pointer directly.

How To Use It

The input and index files must already be opened and have channel numbers. If you input a data file, it must be sorted in ascending order on the key for which you're building an index -- the key can have multiple fields, but you must have specified the locations of the key fields and their sizes. Tab file keys can also have multiple fields. To use IBUILD, you must build a string and send it into the common area with a BLOCK WRITE statement, and then SWAP to IBUILD. The string contents are shown in Table 7-2.

Table 7-2. IBUILD String Contents

Substring Location	Size in Byte	Contents
1,4	4	Channel number of the opened input file.
5,8	4	Byte offset to record 0 of input file.
9,12	4	Maximum number of records in input file.
13,16	4	Number of bytes per record in input file.
17,20	4	Channel number of opened index file.
21,24	4	Byte offset to record 0 of input file.
25,28	4	Maximum number of records in index file (use INDEXCALC to calculate this).
29,32	4	Number of bytes per record in index file, always 512.
33,34	2	Blocking factor (keys per block; use INDEXCALC).
35,36	2	Flag to allow duplicate keys: 1=allow, 0=do not allow.
37,38	2	Flag to check for deleted records: 1=check, 0=do not check.
39,40	2	Total key field length in bytes.
41,42	2	Number of key fields, or -1 (tag file input) or -2 (index file input).
43+	2*2	Starting and ending byte numbers (relative to 1) of key fields in order of significance.

Licensed Material-Property of Data General Corporation

IBUILD (continued)

IBUILD, using STMA 2,1,ERCODE, returns an error value. ERCODE is 0 if the IBUILD is successful, or the appropriate BASIC error code for an error (see Appendix C). Also, IBUILD returns errors in ON ERR traps, sending the line number of the line that caused the error in STMA 2,2,LINENO. The following error codes may occur when you are using IBUILD:

68 index space exhausted.
167 illegal blocking factor.
-52 illegal index file offset (not on a sector boundary).
165 illegal index record size (must be 512).
166 input record out of sequence.

For more information about creating indexes, see Chapter 4.

IBUILD Example

```
0010 DIM X$(512)                    :X$ will hold arguments sent to common.
.                                   :Code to open both files goes here;
.                                   :channel number of input file is C%,
.                                   :channel number of new index file is CI%.
0100 LET X$=CHR$(C%,4),CHR$(0,4)          :Data file has 0 offset,
0105 LET X$(0)=CHR$(100,4),CHR$(100,4)    :100 records, 100 bytes per record.
0110 LET X$(0)=CHR$(CI%,4),CHR$(0,4)      :Index file has 0 offset,
0115 LET X$(0)=CHR$(8,4),CHR$(512,4)      :8 blocks, and 512 bytes per block.
0120 LET X$(0)=CHR$(36,2)                 :Blocking factor is 36 keys per block.
0125 LET X$(0)=CHR$(1,2),CHR$(1,2)        :Allow duplicates , deleted rec. check.
0130 LET X$(0)=CHR$(10,2),CHR$(2,2)       :Total key 10 bytes, 2 fields.
0140 LET X$(0)=CHR$(1,2),CHR$(4,2)        :First field, bytes 1 to 4.
0150 LET X$(0)=CHR$(15,2),CHR$(20,2)      :Second field, bytes 15 to 20.
0160 LET X$(0)=FILL$(0)                   :No more fields.
0170 BLOCK WRITE X$                       :Send into common area.
0180 SWAP "IBUILD"
0184 LET ERCODE=0                         :Initialize ERCODE and LINENO
0186 LET LINENO=0                         :for use with STMA's.
0190 STMA 1,1,ERCODE                      :Get error code, then get line number
0200 STMA 1,2,LINENO                      :of ON ERR trap, if any, from IBUILD.
0210 IF ERCODE THEN GOTO 0700             :If an error, go to 700.
0220 IF LINENO THEN GOTO 0800             :If an error trap, go to 800.
.
.
.
```


Licensed Material-Property of Data General Corporation

Subroutine
• *Utility*

INDEXBLD

Build an index file.

Why Use It?

INDEXBLD is a question and answer utility designed to be RUN or CHAINed, though you can also SWAP to it from your program. INDEXBLD builds an index file from an unsorted data file, tag file, or another index file. You can rebuild an index specifying a blocking factor, and you can choose which method (KADDs or IBUILD) to use to build the index file.

What It Does

If input file is a data file, INDEXBLD calls TBUILD to build a sorted tag file for the data file (which in turn uses QFILESORT). INDEXBLD then either calls IBUILD or uses KADDs (XBUILD) to build the index file (your choice). If the input file is a sorted tag file, INDEXBLD bypasses TBUILD; and when the input file is another index file, a rebuild is assumed using the new specified blocking factor. If the input file is sorted tag or another index, IBUILD is always used to generate the new index file.

How To Use It

You can RUN "INDEXBLD", CHAIN or SWAP to "INDEXBLD", or execute INDEXBLD from the BASIC CLI. The program asks for the input file's filename, whether it is an index, data, or sorted tag file; if it's a data file, it asks you to choose between using KADDs (XBUILD) or IBUILD. Index and tag files are sorted, so IBUILD automatically called for them; for unsorted data files, XBUILD is faster. INDEXBLD looks for a PARAM entry defining the input file, and if not found, INDEXBLD issues a warning and asks for the byte offset, record size, and file size of the input file. If your input file is a data file, INDEXBLD will ask if you want it to check for deleted records. If you do, deleted records (status=0) will be omitted from the intermediate file.

INDEXBLD then asks for the number of fields in the key, and if your input file is a data file, it will ask for the starting and ending byte locations for each field in the key.

Finally, INDEXBLD asks for the logical filename of the index file you are building, and if the file is not defined in PARAM, it asks for the byte offset to the first record in the file. INDEXBLD also asks for the blocking factor, and whether or not duplicate keys are allowed.

To fully understand these questions, you should read Chapter 4.

INDEXBLD Example

```
*RUN "INDEXBLD" )

INPUT FILENAME: CUST )
INPUT FILE IS INDEX(0),DATA(1) OR SORTED TAG(2): 1 )
USE IBUILD(0) OR KADD1: 1 )
CHECK FOR DELETED RECORDS: YES(1),NO(0): 1 )
NUMBER OF FIELDS IN KEY: 2 )
KEY FIELD 1 LOCATION IN DATA RECORD (BYTE 0-N): 0 )
KEY FIELD 1 LENGTH IN BYTES: 4 )
KEY FIELD 2 LOCATION IN DATA RECORD (BYTE 0-N): 4 )
KEY FIELD 2 LENGTH IN BYTES: 6 )
INDEX FILENAME: CUSTINDEX )
DUPLICATE KEYS ALLOWED? YES(1),NO(0): 1 )
*
```


Licensed Material-Property of Data General Corporation

Subroutine
● *Utility*

INDEXCALC

Calculate index information.

Why Use It?

Whenever you have to make a PARAM entry for an index file, or use INDEXBLD or other index building routines, you need to know how many key entries can fit in an index block (record), how many blocks you need in the index file (maximum number of records), and how many keys per block will result from a specified blocking factor. INDEXCALC is designed to be run at your terminal to give you that information immediately.

What It Does

Based on your key entry size, the number of bytes in your data record, the maximum number of data records and a specified blocking factor, INDEXCALC tells you the number of keys per index block, the number of blocks at level 0 and level 1, the number of sectors (blocks) in the index, and the number of sectors (blocks) in the data file.

How To Use It

To run INDEXCALC, use RUN, CHAIN, or SWAP, or execute it from the BASIC CLI. For IBUILD, XBUILD, and TBUILD, you must specify a blocking factor argument (not a percentage), so use the number of keys per index block. For PARAM entries and other routines that ask for it, the maximum number of records in an index file will always equal the number of sectors in the index, which is returned by INDEXCALC.

Example

```
*RUN "INDEXCALC" )

BYTES PER KEY: 10 )
BYTES PER RECORD: 100 )
MAX NUMBER OF RECORDS: 100 )
INDEX BLOCKING FACTOR: 50 )

36  KEYS PER INDEX BLOCK
6   BLOCKS AT LEVEL 0
1   BLOCKS AT LEVEL 1
8   SECTORS IN INDEX
20  SECTORS IN FILE

*
```


Licensed Material-Property of Data General Corporation

Subroutine
● *Utility*

INDEXPRT

Print an index.

Why Use It?

INDEXPRT prints all levels of an index, showing the index entry pointers to other levels, and returns index file information.

What It Does

INDEXPRT requests the logical filename of the index file, and looks it up in the PARAM file; if not found, it assumes you gave it a physical filename and it asks for the byte offset to the first record. INDEXPRT then asks you for the key type (integer, string or crammed), and whether you want the printout to go to the line printer or to your terminal. In the printout, control characters and lowercase characters are printed as decimals (ASCII equivalents) within angle (<>) brackets. Crammed keys are uncrammed without checking for invalid crammed values.

How To Use It

You can RUN, CHAIN, SWAP, or execute INDEXPRT from the BASIC CLI. INDEXPRT supports three key types: integer, string, and crammed. Integer keys must not exceed four bytes.

Examples

```
*RUN "INDEXPRT" )

LOGICAL FILENAME: KEYWORD.TX )
KEY TYPE (1-INTEGER,2-STRING,3-CRAM) ? 2 )
LISTING GOES TO (0)$TRO, (1)$LPT: 1 )
     KEY SIZE =  24
     KEYS/BLOCK =  18
     MAX BLOCKS =  837
     NEXT BLOCK =  37
     BLOCK ZERO =  21
     BLOCKING FACTOR =  72 %
     DUPLICATE KEYS ALLOWED

     BEGINNING KEY:
     BEGINNING KEY = "
     "
     01 @                          3
     01 BLOCK WRITE                1
     01 FM FIND NEXT               3

     02 ABS                        3

     03 ABS                        1

     04 ABS                        1

     05 AND                        3

     06 APPEND                     8
```

```
     07 ASC                        3

     08 ASG                        8

     09 ATTACH                     7

     10 BLDCOM                     8

     11 BLOCK READ                 3

     12 BLOCK WRITE                3

     01 BUILD                      8
     02 CHR$                       19

     02 BYE                        3

     03 CCONT                      8

     04 CDIR                       8

     05 CHAIN                      8

     06 CHAIN                      3

     07 CHATR                      8

     08 CHLAT                      8

     09 CHR$                       3

     01 CLI                        7
     03 DEF                        22

     02 CLOSE                      3

     03 CON                        3

     04 CPART                      8

     05 CRAND                      8

     06 CREATE                     8

     07 CRM$                       3

     08 DATA                       3

     09 DEF                        3

     01 DELAY                      3
     04 DUCAETX                    4

     02 DELETE                     8

     03 DELETE                     3

     04 DELREC.SL                  7

     05 DIM                        3

     06 DIR                        8

     07 DISK                       8

     08 DIV.SL                     7

     09 DOC                        7
```


Licensed Material-Property of Data General Corporation

Subroutine
- *Utility*

INITFILE

Initialize an index or data file.

Why Use It?

Use INITFILE to build record 0 of a linked-available-record file (dynamic allocation) or to build block 0 of an index file. INITFILE also makes a PARAM entry for your file if your file hasn't got one.

What It Does

INITFILE asks you for the logical filename of your file, and asks whether it is an index or a data file. If your file is not in the PARAM file, it asks you if you want to make a PARAM entry, and it asks you for the byte offset to record 0 in the physical file, the record size in bytes, and the last record number (maximum number of records) in the file. INITFILE will then build record 0 of the data file (or block 0 of the index file) allowing you to have linked-available-records in the data file (index files *must* have a block 0). Finally, INITFILE asks if you want to fill the file with nulls.

How To Use It

You can RUN, SWAP, or CHAIN INITFILE, or execute it from the BASIC CLI. To find the maximum number of records (sectors) in an index file, use INDEXCALC. Index records (sectors) are always 512 bytes in length. You should null fill your file if you want to allocate the space for it at this moment, or if the file already exists and you want to get rid of old data. For index files, INITFILE also asks for the number of bytes per key, the blocking factor (percentage) and whether or not duplicate keys are allowed.

For more information about INITFILE, see Chapter 4.

INITFILE Example

INITFILE Example

```
*RUN "INITFILE" )

INDEX(0), DATA(1) OR STOP(2)? 1 )
LOGICAL FILENAME: DOMESTIC )
FILE NOT IN PARAM FILE! DO YOU WISH TO ADD? YES )
     MASTER FILENAME: TOPSECRET )
     BYTE OFFSET TO RECORD 0: 1500 )
     RECORD SIZE IN BYTES: 100 )
     LAST RECORD NUMBER: 10000 )
SHOULD FILE BE NULL FILLED? NO )
INDEX(0),DATA(1),OR STOP(2)? 0 )
LOGICAL FILENAME: DOMINDEX )
FILE NOT IN PARAM FILE! DO YOU WISH TO ADD? YES )
     MASTER FILENAME: TOPSECIND )
     BYTE OFFSET TO RECORD 0: 51200 )
     RECORD SIZE IN BYTES: 512 )
     LAST RECORD NUMBER: 8 )
SHOULD FILE BE NULL FILLED? NO )
BYTES PER KEY? 10 )
BLOCKING FACTOR (% PERCENT): 50 )
DUPLICATE KEYS ALLOWED? YES )
INDEX(0), DATA(1) OR STOP(2)? 2 )

*
```


Licensed Material-Property of Data General Corporation

INITINDEX.SL

Initialize an index file.

Input Variables

Y$ index file descriptor string (see Chapter 4).

X key in length in bytes.

Y flag to allow duplicate keys: 1=duplicate allowed, 0= no duplicates.

Z index file blocking factor (in percent).

X1 file size (in blocks) -- get value from INDEXCALC.

Output Variables

None.

Scratch Variables

T9$ string to hold a block, must be DIMensioned to at least 512 bytes.

Line Numbers

7700 entry point to INITINDEX.SL.

7700-7790 INITINDEX.SL occupies these line numbers.

Why Use It?

INITINDEX.SL is a fast way to initialize an index file from your program. INITINDEX.SL will do an INITFILE for your index file (see INITFILE) by building block 0 (and creating block 1); it does not, however, make a PARAM entry for you.

What It Does

INITINDEX.SL builds block 0 from input arguments, using the file descriptor string (described in Chapter 4), the key length, and the blocking factor, and it sets the duplicate key flag.

How To Use It

To use INITINDEX.SL, you must ENTER it into your program and GOSUB to 7700 after you've set up your input variables. See Chapter 4 for a description of the index file descriptor string. This string contains the byte offset to record 0 in the physical file for this index file, the channel number on which the index file is open, an automatic locking flag, and the logical filename of the index. You must have already opened the index file, so that you can put the channel number in the descriptor string.

INITINDEX.SL Example

```
.                                  :Code to open index file goes here; Y$ is
.                                  :descriptor string, C% is the channel number.
.
0050 LET Y$=CHR$(C%,2),CHR$(0,4)          :Index file opened on C%, 0 offset.
0060 LET Y$(0)=CHR$(0,2),"INDEX",FILL$(0)       :INDEX has auto lock.
0070 LET X=10                              :10 bytes per key.
0080 LET Y=1                               :Allow duplicate keys.
0090 LET Z=50                              :50% blocking factor.
0100 LET X1=8                     :INDEX has 8 blocks-- INDEXCALC returns this.
0110 GOSUB 7700                   :Go to INITINDEX.SL.
.
.
.
```


Licensed Material-Property of Data General Corporation

Subroutine
● *Utility*

LOCKS

Display your current locks.

Why Use It?

Use LOCKS to display the current locks for your job (terminal). Each job has its own unique set of locks identifiers. LOCKS displays the locks you've set, each one's filename and lock area, so that you don't try another LOCK with the same identifier (see LOCK/UNLOCK in Chapter 3, and "File Locking" in Chapter 4).

What It Does

LOCKS displays all the locks you've set that haven't yet been UNLOCKed. During a run of LOCKS, if an UNLOCK occurs during display, the lock associated with the UNLOCK will disappear.

How To Use It

You can RUN "LOCKS", or SWAP or CHAIN "LOCKS", or execute LOCKS from the BASIC CLI. LOCKS does not require any arguments.

Subroutine
● *Utility*

LSPEED

Set your terminal's speed.

Why Use It?

LSPEED allows you to change the speed (baud rate) of a terminal operating on an ALM multiplexor line.

What It Does

LSPEED begins by telling you the line's characteristics and present speed, and it asks you what speed you desire. The baud rate you specify is checked against a table of valid baud rates for this line. If the speed you select is valid, LSPEED tells you to go ahead and change the baud rate (use the knob on the back of the terminal), and then it waits for you to type YES to the query DONE? It waits up to 30 seconds; if you do not respond YES during that time, it resets the original speed. If it receives a response and you correctly adjusted the baud rate, you get a message indicating the new baud rate.

How To Use It

You can RUN "LSPEED", CHAIN or SWAP "LSPEED", or execute LSPEED from the BASIC CLI.

LSPEED Example

```
*RUN "LSPEED" )

YOUR CURRENT LINE CONFIGURATION IS:
8 BITS WITH NO PARITY AND 1 STOP BIT(S)
YOUR CURRENT BAUD RATE IS 1200 BAUD.
CHANGE IT TO? 4800
CHANGE TERMINAL TO 4800 BAUD, AND TYPE "YES" AND A RETURN.
DONE? YES )
YOU'RE RIGHT! IT'S NOW 4800 BAUD.

*
```


Licensed Material-Property of Data General Corporation

MUL.SL

Calculate percent of a number.

Input Variables

PCNT percentage by which Y is multiplied. PCNT is a four place percentage (nn.nn%).

Y numeric expression of value whose percentage is taken.

Output Variables

X value of PCNT of Y.

Scratch Variables

S sign of X.

Z1 temporary value of Y.

Z2 absolute value of PCNT.

Line Numbers

7500 only entry point to MUL.SL.

Why Use It?

Use MUL.SL to find the percent of a number.

What It Does

MUL.SL calculates X as PCNT*Y/10000.

How To Use It

To use MUL.SL, you must ENTER it into your program, assign a value to Y and PCNT, and GOSUB to 7500.

Example

```
*ENTER "MUL.SL" )

0010 LET Y=100
0020 LET PCNT=25
0030 GOSUB 7500
0040 PRINT X
*RUN )
25
*
```


Licensed Material-Property of Data General Corporation

Subroutine
● *Utility*

OPEN

Open physical files (or subfiles).

Why Use It

Given the logical filename of a subfile and a PARAM entry for the logical filename, OPEN opens the physical file in which your logical file (subfile) resides, and returns information with which you can fill the C1 (file characteristics) array. Chapter 4 describes the use of the OPEN utility, the C1 array, and other subfile conventions.

What It Does

OPEN searches the PARAM file for an entry describing your logical filename. Based on the mode of open you supply, OPEN opens the physical file for your logical file on an available channel (it finds the lowest channel not in use). OPEN also returns the information from the PARAM file for you to use with your C1 array. OPEN opens as many files as you send it.

NOTE: OPEN can't open two subfiles in the same physical file in different modes. OPEN chooses the least severe mode of open.

How To Use It

OPEN is designed to be SWAPped to. You pass a string into the common area with a BLOCK WRITE, and SWAP to "OPEN"; then you BLOCK READ the common area for the C1 information. The string you pass to common must be DIMensioned to at least 512 bytes, and should contain the logical filename and the mode of open for each file you want to open, separated by commas. Modes of open are discussed in Chapter 4 and under the OPEN FILE statement in Chapter 3.

OPEN returns the channel number of the opened file in the C1 information passed to common. You retrieve this information with a BLOCK READ into a string, and extract the information (4 bytes per element for C1 array) from the string using the ASC function. GETREC.SL and POSFL.SL would then use the C1 array to position to and get an available record from the logical file.

The string you send to common can include physical files, but they must have PARAM entries for their filenames. Your string cannot have imbedded spaces, and it must be filled with nulls to the DIMensioned length.

OPEN Example

```
0010 DIM X$(512),C1(2,3),REC$(48)          :Record size of SUB2 is 48.
0020 LET X$="SUB1,5,SUB2,5,PHYS,6",FILL$(0)
0030 BLOCK WRITE X$                        :Send logical file info into common.
0040 SWAP "OPEN"                           :OPEN will return X$ with C1 array.
0050 BLOCK READ X$                         :Retrieve info from common area.
0060 LET K=1                               :Pointer to first element in string.
0070 FOR I=0 TO 2                          :For each logical file, 0, 1 and 2,
0080   FOR J=0 TO 3                        :and for each dimension of C1 array,
0090     LET C1(I,J)=ASC(X$(K,K+3))        :extract element, put in C1 array.
0100     LET K=K+4                         :Bump pointer 4 bytes.
0110   NEXT J
0120 NEXT I
.
.
.
```


Licensed Material-Property of Data General Corporation

Subroutine
● *Utility*

PORTS

Display jobs on the system.

Why Use It?

PORTS is a short form of STAT. It displays the accounts that are logged on and an abridged status.

What It Does

PORTS shows you the accounts logged on, the job number and the terminal associated with each job, and an alpha indicator. See the example for further clarification.

Example

```
*RUN "PORTS" )

00 S  C  R  AATLB6  01
01    C  R  ABDGC6  02
03 I  D  C  OPFPG6  03
↑  ↑  ↑  ↑     ↑     ↑
a. b. c. d.    e.    f.
```

Where:

a. The job number of the job.

b. An indicator showing the current I/O status:

I	waiting on input
O	waiting on output
S	waiting on system call
D	waiting on DELAY
L	waiting on LOCK
(blank)	I/O satisfied

c. An indicator showing the current location of the program where:

A	program in core; main partition
M	program in extended memory (hyperspace) not in user partition
D	program is on disk (swapped out)

d. An indicator showing what the current program is doing:

R	program is running
C	program is in compile state
B	program is both running and compiling

e. The account code used to log on.

f. The terminal line number with this job (-1 if the job is de-attached).

Licensed Material-Property of Data General Corporation

POSFL.SL

Position a file pointer to a record.

Input Variables

C1 file characteristics array (see Chapter 4).

F% logical file number (row in C1 array).

R1 record number.

V% byte offset into record (if using entry line 9612).

Output Variables

C% physical file's channel number (from C1 array).

R9 byte position of record R1 in the physical file (useful for rereading or rewriting).

R8 byte position of record in the logical file (useful in LOCK statements).

Line Numbers

9610 entry point to position file to the beginning of record R1.

9612 entry point to position file to the byte offset in record R1 specified in V% (0 is first byte in record).

9610-9645 POSFL.SL occupies these line numbers.

Why Use It?

If you're using subfiles and/or the C1 array, POSFL.SL provides a quick and easy way to position to a record or a byte within a record.

What It Does

POSFL.SL uses information from the C1 array. Your logical file number F% corresponds to the row in the C1 array where the information for the file is kept. POSFL.SL uses R1 to find the record, and if you specify V% as an offset and GOSUB to 9612, POSFL.SL will position the file pointer using V% as an offset into the record; otherwise, entry point 9610 will position the pointer to the beginning of record R1.

How To Use It

To use POSFL.SL, you must ENTER "POSFL.SL" into your program. You must have already set up a C1 array. Information about the C1 (file characteristics) array is in Chapter 4.

If you use the V% offset, R8 will be relative to V%, *not* to the beginning of the logical file, so watch its use in LOCK statements.

POSFL.SL Example

```
                        :Code to open files and fill C1 array goes here.
.
.
.
0130 LET F%=1                       :Logical file 1 (row 1 in C1 array).
0140 LET V%=2                       :Offset 2 bytes from byte 0 (third byte).
0150 INPUT "RECORD NUMBER: ",R1
0160 GOSUB 9612                     :Position to record R1 (using offset V%).
0170 READ FILE (C%),REC$            :C% returned by POSFL.SL.
.
.
.
```


Licensed Material-Property of Data General Corporation

PROTFORM.SL

Protect/unprotect a formatted screen field.

Input Variables

X input value to be output to field (numeric).

This subroutine uses FORM.SL and UNFORM.SL subroutines and requires the same input variables.

Output Variables

E error code (same as FORM.SL and UNFORM.SL).

Scratch Variables

Same as FORM.SL and UNFORM.SL

Line Numbers

9200 converts the field to an unprotected field, and outputs X (or X$) using formatted output.

9250 converts the field to a protected field, and outputs X (or X$) using formatted output.

Why Use It?

PROTFORM.SL converts a protected field to an unprotected field, and an unprotected field to a protected field, depending on the entry point you use.

What It Does

PROTFORM.SL outputs X (or X$ if you are outputting strings) to the field you specify in the format you specified for that field in your screen file.

How To Use It

To use PROTFORM.SL, you must ENTER "PROTFORM.SL", "FORM.SL", and "UNFORM.SL". See these subroutines and Chapter 6 for more information and examples.

Licensed Material-Property of Data General Corporation

Subroutine
- *Utility*

QFILESORT

Fast sort of a file.

Why Use It?

QFILESORT sorts a data file quickly, in ascending order, on a single unsigned alphanumeric key. QFILESORT is faster than FILESORT, but FILESORT sorts on multiple keys and in any kind of collating sequence.

What It Does

QFILESORT sorts a data file in place on the disk. QFILESORT only sorts on a single unsigned binary key, but it does not check for deleted records as FILESORT can.

How To Use It

You must SWAP to QFILESORT from your program and pass arguments through the common area (the same way you use FILESORT). You must open your data file and pass its channel number as an argument. The string of arguments is the same as for FILESORT, except that substring locations 12,12 (sort modifier to check for deleted records) and 18,18 (key type to sort in a different order) must have a zero value.

QFILESORT Example

```
.
.
1000 LET X$=CHR$(0),CHR$(10000,4),CHR$(32,2),CHR$(0,4)
1010 LET X$(0)=CHR$(0),CHR$(1),CHR$(1,2),CHR$(4,2),FILL$(0)
1020 BLOCK WRITE X$
1030 SWAP "QFILESORT"
1040 BLOCK READ X$
1050 IF ASC(X$(1,2)) THEN GOTO 2000
.
.
.
```


Licensed Material-Property of Data General Corporation

Subroutine
• *Utility*

RELINK

Clean up a linked data file.

Why Use It?

If you have a dynamically allocated file (linked-available-record format) whose deleted record chain is destroyed, use RELINK to recreate the deleted record chain.

What It Does

RELINK searches the PARAM file for the logical filename you give it. If a PARAM entry is not found, RELINK asks you for the byte offset in the physical file for record 0, the record size, and the maximum number of records. RELINK then searches for records with a 0 status (first 2 bytes of the record equals 0) and puts them on the new deleted record chain.

How To Use It

You can CHAIN, SWAP, or RUN "RELINK", and it will ask you for the logical filename. You can also execute RELINK from the BASIC CLI by typing:

```
!RELINK filename
```

where filename is the logical file. See Chapter 4 for a description of dynamic allocation (linked-available-records).

Examples

```
*RUN "RELINK" )

LOGICAL FILENAME: CUST )

*
```

or:

```
!RELINK CUST )

!
```

Subroutine
• *Utility*

RNAM

Rename program variables.

Why Use It?

RNAM renames a save file's program variables.

What It Does

RNAM tells you the number of variables and the program name (you give it the name of the save file). RNAM then asks for an old variable name, and you respond with one, and then it asks you for the new name you want to give it, and you respond with one. The program continues like this until you hit the interrupt key.

How To Use It

You can CHAIN, SWAP, or RUN "RNAM", or execute RNAM from the BASIC CLI. RNAM asks for the program name, and you respond with the name with which you SAVEd (or REPLACEd) the program. You can make variable names up to 6 characters: a single letter followed by 1 to 5 alphanumeric characters and ending with nothing (integer), a percent sign (%) (single-precision integer), a number sign (#) (triple-precision integer, if you're using them) and a dollar sign ($) (for strings).

Example

```
*RUN "RNAM" )
PROGRAM NAME: PROG1
PROG1 18 VARIABLES

OLD NAME: X$ )
NEW NAME: STRING$ )
OLD NAME: ikey
*
```


Licensed Material-Property of Data General Corporation

Subroutine
● *Utility*

SCHANS

Display the system channel assignments.

Why Use It?

SCHANS displays the current channel assignments for the entire BASIC system.

What It Does

SCHANS displays the channel assignments in four columns: the first column is the RDOS/DOS channel number, the second is the user status table address, the third is the filename of the physical file opened on that channel, and the fourth is the account ID of the job associated with the file. Channels in use by the BASIC system will not have an ID. See the example.

SCHANS Example

```
*RUN "SCHANS" )

0   32768  BBASIC.OL
1   32768  BASIC.PS
2   32768  BASIC.SW
3   32768  BASIC.ER
4   32768  .
5   32768  .
6   32768  $TTI
7   30469  CUST          AATLB6
8   30894  TOPSECRET     ABDGC6
9   30894  INFORMERS     ABDGC6
10  30469  CUST.TB       AATLB6
11  30469  CUSTINDEX     AATLB6

*
```


Licensed Material-Property of Data General Corporation

Subroutine
● *Utility*

TBUILD
Build a tag file.

Why Use It?

Use TBUILD to build a tag file from a data file or an index file. You can then quickly sort the tag file using QFILESORT.

What It Does

TBUILD constructs a tag file with records consisting of a key and the pointer to the data record. The input file to TBUILD can be an index file or a linked-available-record file with the key imbedded in the data record (keys can have multiple fields). If your input file is a data file, TBUILD skips record 0 and checks the status of each record (first 2 bytes) for a value of 0 (deleted records): deleted records will not be included in the tag file. The record number is computed based on the record accessed (relative to 0, since it skips record 0).

How To Use It

You SWAP to TBUILD after passing arguments into the common area. The arguments are the same as those for IBUILD, except substring location 41,42 (number of key fields), which for TBUILD is the number of key fields or -1 if input is an index file. The arguments in IBUILD that refer to the new index being built are changed in TBUILD to refer to the new tag file being built (e.g., record size does not have to be 512). TBUILD also returns ERCODE using STMA 2,1,ERCODE and LINENO using STMA 2,2,LINENO (see IBUILD).

TBUILD Example

```
0010 DIM X$(512)                        :X$ will hold arguments sent to common.
.                                       :Code to open input and tag files goes here.
.                                       :Channel number of input file is C%,
.                                       :channel number of new tag file is CT%.
0100 LET X$=CHR$(C%,4),CHR$(0,4)          :Input file has 0 offset,
0105 LET X$(0)=CHR$(100,4),CHR$(100,4)    :100 records, 100 bytes per record.
0110 LET X$(0)=CHR$(CT%,4),CHR$(0,4)      :Tag file has 0 offset,
0115 LET X$(0)=CHR$(8,4),CHR$(512,4)      :8 blocks, and 512 bytes per block.
0120 LET X$(0)=CHR$(36,2)                 :Blocking factor is 36 entries/block.
0125 LET X$(0)=CHR$(1,2),CHR$(1,2)        :Allow duplicates , deleted rec. check.
0130 LET X$(0)=CHR$(10,2),CHR$(2,2)       :Total key 10 bytes, 2 fields.
0140 LET X$(0)=CHR$(1,2),CHR$(4,2)        :First field, bytes 1 to 4.
0150 LET X$(0)=CHR$(15,2),CHR$(20,2)      :Second field, bytes 15 to 20.
0160 LET X$(0)=FILL$(0)                   :No more fields.
0170 BLOCK WRITE X$                       :Send into common area.
0180 SWAP "TBUILD"
0182 BLOCK READ X$                        :Retrieve information from TBUILD.
0184 LET ERCODE=0                         :Initialize ERCODE and LINENO
0186 LET LINENO=0                         :for use with STMA's.
0190 STMA 1,1,ERCODE                      :Get error code, then get line number
0200 STMA 1,2,LINENO                      :of ON ERR trap, if any, from TBUILD.
0210 IF ERCODE THEN GOTO 0700             :If an error, go to 700.
0220 IF LINENO THEN GOTO 0800             :If an error trap, go to 800.
.
.
.
```


Licensed Material-Property of Data General Corporation

Subroutine
● *Utility*

TERM

Change certain terminal key functions.

Why Use It?

Use TERM if you want to change the de-attached key, the line cancel key, the character delete key, the character delete echo character, and the interrupt key.

What It Does

TERM asks you for a value for each key. You hit the actual key you want to change it to, and TERM echoes the ASCII value of the key you hit.

How To Use It

You can CHAIN, SWAP, or RUN "TERM", or execute TERM from the BASIC CLI. Answer each question with the actual key you want to use for the function. In our example, we hit CTRL-D for the de-attach key, CTRL-X for the line cancel key, CTRL-H for the delete key and the delete character echo, and NEW LINE for the interrupt key.

Example

```
*RUN "TERM" )

DE-ATTACH CHAR: (ctrl-d) 4
LINE CANCEL CHAR: (ctrl-x) 24
CHARACTER DELETE CHAR: (ctrl-h) 8
CHARACTER DELETE ECHO: (ctrl-h) 8
IKEY CHAR: (new line) 10
*
```

Subroutine
● *Utility*

UCHANS

Display your channel assignments.

Why Use It?

Use UCHANS to find out what files you have opened on which channels. Each job (terminal) can have up to 15 channels in use; UCHANS tells you what you've used already at your terminal.

What It Does

UCHANS displays the current channel assignments for your job. It displays two columns: the first is the channel number, and the second is the physical filename opened on that channel. See the example.

Example

```
*RUN "UCHANS" )

0 CUSTLOG.LG
1 CUST
2 $LPT
5 CUSTI3
7 CUSTI2
8 CUST.TB

*
```


Licensed Material-Property of Data General Corporation

UNFORM.SL

Unformatted screen input/output.

Input Variables

E error code, initialized to 0.

F numeric expression for the field number. F equals the field number if R equals the row, or F equals 100*row number plus the field number.

R numeric expression for row number if F < 100.

X$ string DIMensioned to at least 80 bytes; value to be input from screen field or output to screen field.

SCRN numeric expression for the screen number in the screen file.

Output Variables

E error code.

FUNC function number for a function key.

X$ string to receive input or send output to screen fields.

Scratch Variables

LINE$ used for input and output to CRT, must be DIMensioned to at least 132 bytes.

SCRN$ describes each field in the screen file created by SM. Must be DIMensioned to at least 48 bytes plus three times the number of fields on the screen.

XCOL column number of the current field.

XDEC number of digits past the decimal point (used only in formatted I/O -- FORM.SL).

XFLGS bit flags describing the field as follows:

Bit value	Meaning
128	Set if it is a numeric field, else it's a string.
64	Set if it is a protected field, else it's unprotected.
32	Set if the field is to be left justified.
16	Set if a numeric field may have varying numbers of decimals on input.
8	Set if a numeric field may be negative on input.
4	Set if a string field is crammed.
2,1	Set to XDEC if field is numeric.

XLROW number of last line read from screen (indicates the contents of LINE$).

XROW line number of the current field.

XWID width of the current field.

Line Numbers

9100-9895 UNFORM.SL, FORM.SL, and PROTFORM.SL occupy these line numbers. (See FORM.SL).

9150 Sets the necessary STMA's, block mode, and locks the keyboard; always used.

9155 Same as 9150, but used if lowercase is not needed.

9175 Resets the STMA's set by 9150 or 9155, turns interactive mode back on and clears the CRT screen.

Licensed Material-Property of Data General Corporation

UNFORM.SL (continued)

9400 Reads a file from the screen and returns it in X$. Use 9650 and F (or F and R) to position to the screen field first. If the line is already in LINE$ (known by XLROW), screen field will not be read. XLROW is reset whenever you output a field or allow the user to input a field, and whenever LINE$ is destroyed. Trailing spaces are truncated.

9550 Outputs X$ to the field requested, left justifies, and handles screen overflow.

9650 Positions the cursor to the requested field.

9700 Displays screen number SCRN from a screen file opened on channel 15 and loads SCRN$.

9735 Similar to 9700 except screen is not displayed (i.e., re-read SCRN$ only).

9750 Reads a function key input from the keyboard and returns a value in FUNC: 1-8 for F1 to F8, 14-16 for F9 to F11, -1 if input was not a function key.

9800 Converts a field in F, or in F and R if F < 100, into XROW, XCOL, XPOS, XWID, XFLGS, and XDEC.

9850 Outputs an error message at row 24 of the screen. If E<>0 then it uses the error code in E with the BASIC.ER file to return the appropriate error message. If E=0, then your own message in X$ is output.

9890 Clears the error message from the screen.

Why Use It?

Use UNFORM.SL to input fields from a screen in a screen file or output fields to a screen in a screen file, or to perform screen functions. See Chapter 6 for a complete discussion of screen handling.

How To Use It

To use UNFORM.SL, you must ENTER "UNFORM.SL" into your program. Use each entry point list above to perform each unique function. You can specify a field in two ways: either set R = row number and F = field number, or set F = row number * 100 plus the field number. For a complete description and examples, see Chapter 6.

Licensed Material-Property of Data General Corporation

VAL.SL

Convert a string to a number.

Input Variables

X$ string of digits and certain special characters, depending on the entry point you use.

Output Variables

X signed numeric value of X$.

Y% error termination code (0 if the conversion ended with a comma, space, or end of string; otherwise, contains the ASCII value of the terminating character).

Z% implied decimal point location.

Scratch Variables

I% character pointer.

X% sign of X.

Line Numbers

7600-7659 VAL.SL occupies these line numbers.

7600 Allows a plus sign (+), minus sign (-), and decimal point (.) as valid characters in X$. Z% will be negative if no decimal is found; otherwise, Z% will contain the power of 10 showing the implied location of the decimal point in X.

7608 Same as 7600 except decimal points are not allowed in X$; conversion will stop on a decimal point (Z% is meaningless).

7610 Unsigned integer conversion. Plus sign (+) allowed in X$, but conversion will stop on a minus sign (-) or decimal point (.) (Z% is meaningless).

Why Use It?

VAL.SL converts a string of digits into a number. If you want to make sure the number is not decimal or negative, VAL.SL can stop converting if either is encountered when you use the entry point 7610 (7608 for decimal error only).

How To Use It

To use VAL.SL in your program, you must ENTER "VAL.SL" into your program. You can create a string of digits by assigning the digits to the string (do not use CHR$).

Example

```
0010 DIM X$(6)
0020 LET X$=12345
0030 GOSUB 7610
0040 PRINT X
0050 LET X$="-123.4"
0060 GOSUB 7600
0070 PRINT X/Z%
*RUN )

12345
-123
*
```


Licensed Material-Property of Data General Corporation

Subroutine
• *Utility*

VAR

List the variables in a program.

Why Use It?

VAR supplies a sorted list of variables in any SAVEd program.

What It Does

VAR looks up all variables defined in the symbol table of the save file and sorts the names into alphabetical order. VAR also asks for a file to output the list to; if you don't specify one, it displays the list on your terminal.

How To Use It

You can CHAIN, SWAP, or RUN "VAR", or execute VAR from the BASIC CLI. VAR will ask for the output file -- you can specify the line printer, any output file, or answer with a carriage return and VAR will display the list at your terminal. VAR then asks for the program name, and you must specify the name of a save file (i.e., the name you used to SAVE or REPLACE the program).

Example

```
*RUN "VAR" )

OUTPUT FILE: )
PROGRAM NAME: PROG1 )

PROG1   5 VARIABLES

A$(512)  REC$(128)  R1  C%
C1(2,3)

*
```

Subroutine
• *Utility*

XBUILD

Build an index from a data file.

Why Use It?

XBUILD uses KADDs to build an index from a data file. This method is faster if the data file is not sorted.

What It Does

XBUILD uses KADDs which usually builds an index with approximately a 75% blocking factor; the blocking factor argument is ignored. XBUILD looks for a key in the data file based on the locations you specify and computes the record pointer based on the record access relative to 0. XBUILD expects to receive a linked-available-record file (dynamic allocation). It therefore skips record 0 and checks the status of records (first 2 bytes) so that it doesn't include keys for deleted records (if you specify a deleted record, check the arguments to XBUILD).

How To Use It

To use XBUILD, you build a string exactly like the one for IBUILD and pass it into the common area. The key in your data file can have multiple fields. The input and index files must be opened and their channel numbers passed as arguments. XBUILD also returns errors in the same way as IBUILD. See IBUILD for a description of the arguments you pass to common and for examples (or see Chapter 4).

End of Chapter

Licensed Material-Property of Data General Corporation

Chapter 8
BASIC CLI Commands

The BASIC CLI is a utility program that provides a working CLI for all terminals in a multiuser system. The BASIC CLI is patterned after the RDOS/DOS CLI, but some BASIC CLI commands work differently from their counterpart RDOS/DOS commands, and the BASIC CLI also has additional commands. To use the BASIC CLI, you can either RUN "CLI", SWAP to "CLI", or CHAIN "CLI" (see the CLI utility in Chapter 7).

The BASIC CLI builds the equivalent of a COM.CM file in the common area for any command it does not recognize and swaps to the filename which is the first element of the command. The format is identical to the RDOS COM.CM file except that it can only be 512 bytes long (the common area only allows 512 bytes). You can write your own BASIC CLI command by writing a program that uses GETCM.SL (see GETCM.SL in Chapter 7) to interpret the information passed in the common area.

You can use the RDOS/DOS asterisk (*), dash (-), and indirect (@) file conventions. You can only use the indirect (@) file reference at one level; e.g., @filename@ may not occur in a file being used as an indirect file. The RDOS/DOS use of parentheses for command replication with different arguments is implemented, but no nesting of indirects (@) and parentheses is allowed. If you want to use these conventions, you should first read about their uses in the *RDOS/DOS CLI User's Manual.*

You can call the BASIC CLI from your program in a manner similar to calling the RDOS CLI. You pass the command line to the common area preceded by the string "CLI.CM<0>". The BASIC CLI always reads first seven bytes in the common area looking for this string. If it is found, the remainder of the common area is treated as a series of commands separated by semicolons. In order to have the CLI return to the calling program, you must make your last command in the command list POP or QUIT. You can continue a command line after the end of a line by typing an up arrow (↑) before typing the next line. The size of your command line, however, is limited to the size of the common area. See GETCM.SL in Chapter 7 for a definition of the COM file.

As a normal user, you can't execute the BASIC CLI commands -- CDIR, CHATR, CHLAT, CPART, EQUIV, SDIR, SPDIS, SPEBL, and SPKILL. If you are the account manager, you can use CHATR and CHLAT, but you can't execute the other commands. A system programmer and a system operator can use any BASIC CLI command except SDIR. The system manager can execute all Business BASIC commands and use all features.

In our formats for BASIC CLI commands, we include global and local switches. Use all global switches with the command itself, and use all local switches with the arguments specified for the local switch. This is the same convention used for the RDOS/DOS CLI.

Licensed Material-Property of Data General Corporation

- *RDOS*
- *DOS*

APPEND

Append a file to another file.

Format

APPEND appendfile oldfile$_1$ *[oldfile$_2$ ...]*

Arguments

appendfile The file you end up with, containing the concatenated files. APPEND creates this file unless you use the /N global switch.

oldfile The file you want to append to appendfile. You can have several oldfiles that are concatenated in the order in which you put them as arguments to APPEND. The oldfiles are unchanged.

Global Switches

/N Do not create appendfile: it must already exist or you'll get an error message.

/R Organize appendfile randomly. Without this switch, appendfile is organized sequentially.

Why Use It?

APPEND is useful for appending several small programs to a large main program, or concatenating data, index files, subfiles, etc.

What It Does

APPEND concatenates the files in the order you put them as arguments, and appends them to appendfile (if appendfile exists), or creates a new file appendfile (unless you use the /N switch) containing the concatenated files. Appendfile is organized sequentially unless you specify random organization with the /R switch. Append the switches to the keyword APPEND, as shown in the example.

APPEND Example

```
!APPEND/N MAIN.SR SUB1.SR SUB2.SR FORM.SL VAL.SL UNFORMDG.SL )
```

This appends the programs SUB1.SR, SUB2.SR, and the subroutines FORM.SL, VAL.SL, and UNFORMDG.SL to MAIN.SR in that order. MAIN.SR must already exist, or this causes an error (because we use the /N switch).

Licensed Material-Property of Data General Corporation

- *RDOS*
- *DOS*

ASG

Assign a device for exclusive use.

Format

ASG filename

Arguments

filename The name of a reserved file.

Switches

None.

Why Use It?

Use ASG to assign a specified reserved file (usually a device) to the current job so that you can set up a device (before opening the device in a BASIC program) without having another job open the device first.

What It Does

ASG assigns filename to the job executing ASG until the job logs off or until filename is FREE (see FREE in this chapter).

Example

```
!ASG $LPT )
```

Assign the line printer to this job for exclusive use.

- *RDOS*
- *DOS*

BLDCOM

Build a documentation file for PRTCOM.

Format

BLDCOM filename$_1$ *[filename$_2$ ...]*

Arguments

filename The name of a program source file or listing file in character (ASCII) format. If it has a .FR extension, it is recognized as a FORTRAN source file. You can use dash (-) and asterisk (*) conventions.

Global Switches

/V Display list of filenames on terminal.

Why Use It?

Use BLDCOM to build a documentation file used by PRTCOM if you want to print selected comments from program source files. Your program source files can contain BASIC, FORTRAN, or assembly language programs.

What It Does

BLDCOM requests the name of the documentation file (output of BLDCOM). Comments in a source file or listing file are recognized by the colon (:) for BASIC comments, a semicolon (;) for assembly language program comments, or a C for FORTRAN comments. A comment section beginning with a comment symbol and an exclamation point (!) as the second character will be saved in the documentation file and printed by PRTCOM. Terminate the section with a double comment symbol (e.g., ::, ;;, CC, etc.). The exclamation point comment is interpreted as the name of a program or subroutine section. See PRTCOM for more editing conventions concerning comments.

Example

```
!BLDCOM/V -.SL *BUILD.SR )
DOCUMENTATION FILE: TMP )
.
.
.
!
```

Build a documentation file, TMP, containing the comments for all subroutine library modules (-.SL) and any programs with 6-character names whose last 5 letters are BUILD and have the .SR extension. A verification list of these files will appear at the terminal.

Licensed Material-Property of Data General Corporation

- *RDOS*
- *DOS*

BUILD

Build a filename file (command file).

Format

BUILD outputfile *[filename$_1$ [filename$_2$ ...]]*

Arguments

outputfile A new file you end up with, containing filenames.

filename The name of a file in the current directory. If you want a file in another directory, you must establish a link to it first (if you can). You can use RDOS/DOS dash (-) and asterisk (*) conventions.

Global Switches

/A Include all files, permanent as well as nonpermanent. You need not have *filenames* with this switch.

/E Do not include extensions to filenames.

/K Do not include links.

/N Include only links.

/P Sort list by date of last file access (last open).

/S Sort list alphabetically.

/W Sort list by time of last write to file.

Local Switches

mm-dd-yy/A Include only files created on or after this date, where date is expressed as mm-dd-yy (e.g., 04-13-78).

mm-dd-yy/B Include only files created before this date, where date is expressed as mm-dd-yy.

filename/N Do not include files matching this filename.

Why Use It?

Use BUILD if, for example, you want an alphabetical list of all the files in your directory. Just type BUILD/A/S for all files, permanent and nonpermanent. You append global switches to the keyword BUILD or to another global switch. Local switches, in this case, must take the form of a separate argument. See the examples.

Examples

```
!BUILD TMP -.- )
```

This produces file TMP containing the names of all nonpermanent files in the current directory.

```
!BUILD/A/N/P BUILDLINKS )
```

This produces file BUILDLINKS containing the names of all permanent and nonpermanent files that are alias links to files in other directories. The files are sorted by date of last access, starting with the file opened most recently.

```
!BUILD/K LISTING 02-04-78/B TMP-.-/N )
```

This produces file LISTING containing all nonpermanent files (except links) that were created before 2-4-78 in your directory, but not containing any file that begins with TMP.

Licensed Material-Property of Data General Corporation

- *RDOS*
- *DOS*

CCONT

Create a contiguous file.

Format

CCONT filename$_1$ blocks$_1$ *[filename$_2$ blocks$_2$ ...]*

Arguments

filename The name of file you want to create, must follow filename conventions described in Chapters 2 and 4.

blocks The number of contiguous blocks you need for file length, where a block is 512 bytes (so your file length is blocks *512). You must specify the number of contiguous blocks you need.

Switches

None.

Why Use It?

In order to create a contiguous file, you *must* use CCONT; there is no other way. You can create more than one file in a CCONT command.

What It Does

No attributes other than C are set for the file(s). CCONT reserves the number of contiguous blocks you specify in blocks.

For more information about contiguous files, see Chapter 4.

Examples

```
!CCONT TOM 25 )
```

This creates file TOM as contiguous with a length of 12800 bytes (25 contiguous blocks).

```
!CCONT MASTER 200 NBASIC:TMP 22 )
```

This creates MASTER as a contiguous file of 200 blocks (102400 bytes), and creates a file TMP in directory NBASIC with a length of 22 blocks (11264 bytes).

- *RDOS*
- *DOS*

CDIR

Create a subdirectory (RDOS) or directory (DOS).

Format

CDIR (sub)directoryname

Arguments

(sub)directoryname The name of (sub)directory you want to create.

Switches

None.

Why Use It?

Use CDIR to create a directory in DOS or a subdirectory in RDOS.

What It Does

It attaches a .DR extension to your (sub)directoryname and creates it in the current or given partition. For more information about (sub)directories and partitions, consult the *RDOS/DOS CLI User's Manual.*

Examples

```
!CDIR TONY )
```

This creates subdirectory TONY.DR in the current directory.

NOTE: The current directory in this case must be a primary or secondary partition and not a subdirectory.

```
!CDIR DP0:TOM:DEB )
```

This creates a subdirectory DEB.DR in the secondary partition TOM.DR of the primary partition DP0.

Licensed Material-Property of Data General Corporation

- *RDOS*
- *DOS*

CHAIN

Execute a program.

Format

CHAIN filename

Arguments

filename A string literal (without quotes); the name of a RDOS/DOS save file (in compile format).

Switches

None.

Why Use It?

Use CHAIN to execute a program so the CLI will not come up again when the program is finished (i.e., your program replaces the CLI program).

What It Does

CHAIN overwrites the CLI with filename and executes filename. When filename terminates, you're put back in BASIC keyboard mode (*not* the BASIC CLI, so if you want to use the CLI again, you have to RUN "CLI").

Example

```
!CHAIN "MYPROG" )
```

MYPROG replaces CLI and executes.

- *RDOS*
- *DOS*

CHATR

Change a file's attributes.

Format

CHATR filename$_1$ attrib$_1$ *[filename$_2$ attrib$_2$ ...]*

Arguments

filename A string literal (not in quotes) for a filename in your directory.

attrib One or more of the following attributes:

- N No linking is permitted to this file.
- P Permanent file, can't be deleted or renamed.
- R Read protected file, can't be read.
- S Save file; once set, this attribute cannot be removed.
- W Write protected, can't be written to. This attribute can't be removed from a file that also has the S (saved) attribute.
- 0 Remove existing attributes.
- ? User attribute.
- & User attribute.
- * Retain existing attributes.

Licensed Material-Property of Data General Corporation

CHATR (continued)

- *RDOS*
- *DOS*

Switches

None.

Why Use It?

Use CHATR to change existing attributes, remove existing attributes or add new attributes to a file in your directory. File attributes are described in Chapter 4.

What It Does

CHATR modifies the existing attributes of a file, or adds attributes to a file. It modifies only resolution access attributes. (Use CHLAT to modify link access attributes). You can only use resolution files with CHATR. If a file has the A attribute, its attributes cannot be changed.

Examples

```
!CHATR ABC *PW XYZ 0 )
```

Add the permanent and write protected attributes to the file ABC without removing existing attributes. Remove current attributes from file XYZ.

CHLAT

Change a file's link access attributes.

Format

CHLAT filename$_1$ attrib$_1$ *[filename$_2$ attrib$_2$ ...]*

Arguments

filename A string literal (without quotes) for a filename of a file in your directory.

attrib One or more of the following attributes:

- N Do not allow linking to this file (actually, the link can be created, but not used).
- P Permanent file, can't be deleted or renamed.
- R Read protected file, can't be read.
- S Save file; once set, this attribute can't be removed.
- W Write protected, can't be written to; this attribute can't be removed from a file with the S (save file) attribute.
- 0 Remove existing attributes.
- ? User attribute.
- & User attribute
- * Retain existing attributes.

Licensed Material-Property of Data General Corporation

CHLAT (continued)

- *RDOS*
- *DOS*

Switches

None.

Why Use It?

Use CHLAT to modify or add attributes to a resolution file that will regulate link access. Users linking to this file can only do what you allow them to do with link attributes.

What It Does

CHLAT removes, modifies, or adds link access attributes for a resolution file used to regulate link access from other directories. If a file has the A attribute, its attributes can't be changed. Attributes are described in Chapter 4.

Example

```
!CHLAT TAPEUTIL *PW )
```

Add the permanent and write protected attributes to TAPEUTIL without changing existing attributes; this will allow another user in another directory to link to TAPEUTIL and to execute TAPEUTIL but not to write to TAPEUTIL and not to delete or rename TAPEUTIL.

CPART

Create a secondary partition.

Format

```
CPART partname blockcount
```

Arguments

pathname — The name of secondary partition that you want to create.

blockcount — The number of contiguous blocks, must be greater than or equal to 48 and be a multiple of 16. If your blockcount is not a multiple of 16, it will be truncated to the next lower multiple of 16.

Switches

None.

Why Use It?

Use CPART to create a secondary partition within a primary partition. See Chapter 4 for a description of partitions.

What It Does

CPART creates a secondary partition partname and adds a .DR extension to partname. Your secondary partition is blockcount number of contiguous blocks in length, greater than or equal to 48 blocks and a multiple of 16.

Example

```
!CPART DP0:PART01 50 )
```

Creates a secondary partition PART01.DR in partition DP0; the size of PART01.DR will be 48 contiguous blocks.

Licensed Material-Property of Data General Corporation

- *RDOS*
- *DOS*

CRAND

Create a random file.

Format

CRAND filename$_1$ *[filename$_2$ ...]*

Arguments

filename A string literal (without quotes) for a new filename in your directory you want to be a random file.

Switches

None.

Why Use It?

Use CRAND to create a random file in your directory or a given directory; you can create many files in one CRAND command. Random files are discussed in Chapter 4.

What It Does

CRAND creates a randomly organized file in your directory or in the directory specified. Each random file will have the D attribute and a length of 0; it will grow as required during use.

Examples

```
!CRAND RFILE MYFILE ABC:TMP )
```

Create random files RFILE and MYFILE in your directory, and random file TMP in directory ABC.

- *RDOS*
- *DOS*

CREATE

Create a sequential file.

Format

CREATE filename$_1$ *[filename$_2$ ...]*

Arguments

filename A string literal (without quotes) for the name of your new sequential file.

Switches

None.

Why Use It?

Use CREATE to create an RDOS sequential file.

What It Does

CREATE creates one or more sequentially organized RDOS files in a given directory with 0 length and no attributes. Sequential files are discussed in Chapter 4.

Example

```
!CREATE TMP TMP1 ABC:TEMP1 )
```

Creates files TMP and TMP1 in your directory and another file TEMP1 in directory ABC.

Licensed Material-Property of Data General Corporation

- *RDOS*
- *DOS*

DELETE

Delete a file, directory or partition.

Format

DELETE filename$_1$ *[filename$_2$ ...]*

Arguments

filename A string literal (without quotes) for the name of a resolution file, link file, directory, or partition.

Global Switches

/C Confirm each deletion. The system repeats each filename, then waits for you to confirm the deletion by typing a carriage return. To prevent deletion, type any other key but carriage return.

/L List deleted files on $LPT (overrides /V).

/P Delete files in order of date last opened.

/S Delete files in alphabetical order.

/V Verify deleted files with a list at your terminal.

/W Delete files in order of time of last write.

Local Switches

mm-dd-yy/A Delete only files created on or after the date in the form mm-dd-yy.

mm-dd-yy/B Delete only files created before this date, in the form mm-dd-yy.

filename/N Do not delete files matching the filename.

Why Use It?

Use DELETE to delete files, directories, and partitions. Use it also with a link name to delete the resolution file.

What It Does

DELETE deletes the files named in any directory specified (the directory must be initialized). To delete a directory or partition, you must first RELEASE it. If filename is a link entry, DELETE will delete the resolution file (if its link attributes allow it), *not* the link entry (use UNLINK to delete a link entry). You can use asterisk (*) and dash (-) conventions.

Example

```
!DELETE/V TMP-.** XYZ )

DELETED TMP.SR
DELETED TMP1.SR
DELETED TMP101.SV
DELETED XYZ
```

Deletes all files beginning with TMP having a 2-character extension, and file XYZ, from your directory, displaying the filenames of deleted files on the terminal.

Licensed Material-Property of Data General Corporation

- *RDOS*
- *DOS*

DIR

Change the current directory.

Format

DIR { partname:directory / directory }

Arguments

partname The name of secondary partition if the directory you want resides in another secondary partition.

directory The name of the directory you want.

Switches

None.

Why Use It?

Use DIR to change the current directory (i.e., move to another directory). Directories are discussed in Chapter 4.

What It Does

Changes the current directory to directory or to partname:directory. The current system directory cannot be changed.

Examples

```
!DIR USER )
```

Change the current directory to USER.DR.

```
!DIR PART01:USER )
```

Change the current directory to another USER.DR in partition PART01.DR.

- *RDOS*
- *DOS*

DISK

Display the amount of disk space that was used and what is left.

Format

DISK

Switches

None.

Why Use It?

Use DISK to display a decimal count of the number of sectors (blocks) and bytes used and the number still available in the current partition, as well as the number of contiguous sectors (blocks) still available.

What It Does

DISK returns the information for the current partition (directory); if you are in a subdirectory (or secondary partition), DISK returns the information for the parent partition.

Example

```
!DISK )
```

	USED	*LEFT*	*CONTIGUOUS*
SECTORS	*73,747*	*113,661*	*89,086*
BYTES	*37,758,464*	*58,194,432*	*45,612,032*

This indicates that 113,661 blocks of the 187,408 blocks in your directory are still available for use. Of these, the largest contiguous group is 89,086 blocks.

Licensed Material-Property of Data General Corporation

- *RDOS*
- *DOS*

DUMP

Copy one or more disk files in DUMP format to an output file.

Format

DUMP outputfile *[filename$_1$ filename$_2$ ...]*

Arguments

outputfile The name of file or device to receive dump.

filename An optional file you want to be dumped; only files matching *filename* are dumped. (You can use dash (-) and asterisk (*) conventions.)

Global Switches

/A Permanent files as well as nonpermanent files will be dumped; otherwise, only nonpermanent files will be dumped.

/K Do not dump links.

/L List dumped filenames on $LPT (overrides /V).

/N Dump only links.

/P Dump in sorted order by date of last access.

/S Dump in sorted order alphabetically.

/V Verify dump by listing filenames on the terminal.

/W Dump in sorted order by time of last write.

/X Dump in sorted order by filesize, ascending.

Local Switches

mm-dd-yy/A Dump only files created on or after the date mm-yy-dd.

mm-dd-yy/B Dump only files created before the data mm-dd-yy.

filename/N Do not dump files matching the name filename.

Why Use It?

DUMP is a very reliable way to back up files. You can reload them using LOAD. FDUMP and FLOAD dump and load in a different format, so you should not mix DUMPs with FDUMPs.

What It Does

DUMP copies files from the current directory onto a given file or device (outputfile). If you specify *filename(s)*, only files matching *filename(s)* are dumped. If you omit *filename(s)*, all nonpermanent files in the current directory are dumped. Your outputfile can be a disk file, or it can be a mag tape, cassette, or paper tape file. DUMP outputs using *dump format*, which is described in Appendix C of the *RDOS/DOS CLI User's Manual.*

Examples

```
!DIR DP0 )
```

Change current directory to parent partition DP0.

```
!DUMP/A MT0:0 )
```

Dump all permanent and nonpermanent files to mag tape MT0:0.

```
!DUMP/K SOURCE.DP -.SR 2-1-78/A )
```

Dump all files in the current directory (except links) that have the .SR extension and were created on or after February 1, 1978 to the output file SOURCE.DP.

Licensed Material-Property of Data General Corporation

• *RDOS*
DOS

EQUIV

Rename a device.

Format

EQUIV newname oldname

Arguments

newname The new name you want to give the device temporarily.

oldname The name of a device, or a newname assigned in a previous EQUIV.

Switches

None.

Why Use It?

You use EQUIV to assign a temporary name to a device other than its own name. EQUIV can give your programs device-independence; you can write a generic device specifier in your programs, and EQUIV the device to that generic name at runtime.

What It Does

EQUIV assigns a newname to the device only if the device has not yet been initialized. You cannot EQUIV a master disk device name. The newname will be in use until you RELEASE the device by its newname or until you EQUIV the device again. A RELEASE restores the original device name (oldname). If you EQUIV a device to a newname, the newname is in effect throughout the system.

Examples

```
!EQUIV MYTAPE MT0 )
```

MT0 is now called MYTAPE.

```
!INIT MT0 )
FILE DOES NOT EXIST:MT0
!INIT MYTAPE )
.
.   Throughout the system, MT0 is now MYTAPE.
.
!RELEASE MYTAPE )
!INIT MYTAPE )
FILE DOES NOT EXIST:MYTAPE
!EQUIV MYTAPE MT0 )
!INIT MYTAPE )
```

• *RDOS*
DOS

FDUMP

Fast dump of files to mag tape.

Format

FDUMP magtapefile *[filename$_1$ filename$_2$ ...]*

Arguments

magtapefile The name of a tape file in the form MTn:file, where n is the drive number and file is the tape file number; or the EQUIValent name of a tape drive.

filename The name of any file in the current directory, used as a match; you can use asterisk (*) and dash (-) conventions.

Global Switches

/A Dump permanent as well as nonpermanent files.

/B Dump small tape blocks for compatibility with unmapped systems so that you can use FLOAD to load them onto an unmapped system.

/E Append dump to end-of-tape file.

/I Ignore tape read errors and SYS.DR errors.

/K Do not dump links.

/L List dumped filenames on $LPT (overrides /V).

/M Request user input label and add label to tape.

/N Dump only links.

/V Verify dump by listing filenames on the terminal.

Local Switches

mm-dd-yy/A Dump only files created on or after this date, where date is in the form mm-dd-yy.

mm-dd-yy/B Dump only files created before this date, where date is in the form mm-dd-yy.

filename/N Do not dump files matching the name filename; you can use asterisk (*) and dash (-) conventions.

Licensed Material-Property of Data General Corporation

FDUMP (continued)

- *RDOS*
- *DOS*

Why Use It?

FDUMP is much faster than DUMP because it writes tape blocks of up to 8192 bytes (6144 bytes with the /B switch). You cannot, however, LOAD a tape you FDUMPed, you can only use FLOAD. Note that FDUMP with a /B switch will write smaller blocks for compatibility with unmapped BASIC systems.

What It Does

FDUMP dumps all files in the current directory, or the files specified in a match for *filename* and/or specified by switches, to a given mag tape file, magtapefile. If magtapefile has not been previously INITed, FDUMP will automatically INIT the tape drive and RELEASE the tape drive when it's finished. If the amount of data exceeds the capacity of the tape, FDUMP will request the mounting of additional tape volumes. If you use the /M switch, FDUMP will request an input label for the tape, and the label you give it will be displayed when the tape is reloaded.

Examples

```
!FDUMP/A MT0:0 )
```

Dump all files in the current directory to mag tape unit 0:file 0.

```
!FDUMP/K MT1:4 -.SR 2-1-78/A )
```

Dump all nonpermanent files with the .SR extension created on or after February 1, 1978 to the file MT1:4 excluding links.

FILCOM

Compare two files.

Format

FILCOM filename$_1$ filename$_2$ *[filename/L]*

Arguments

filename$_1$ and filename$_2$	Two files you want compared.

Global Switches

/A Print address in octal and data in character (ASCII) format.

/D Print all numbers in decimal.

/H Print all numbers in hexadecimal.

/L Print output to $LPT.

Local Switches

filename/L Send output to *filename*.

Why Use It?

Use FILCOM to compare two files word by word. File organization of both files may differ.

What It Does

FILCOM compares two files word by word and prints dissimilar word pairs. The displacement is printed, followed by a slash, followed by the dissimilar word pair. It prints in octal unless you specify a global switch, and it prints to the terminal unless you specify /L or *filename/L*.

Examples

```
!FILCOM YIN YANG COMPARE.LS/L )
```

Compare YIN and YANG, outputting the dissimilar word pairs to COMPARE.LS.

Licensed Material-Property of Data General Corporation

- *RDOS*
- *DOS*

FILES

Display names of files in current directory.

Format

FILES

Why Use It?

FILES provides a quick listing of all nonpermanent files in your current directory, showing the filename and L, S, or blank to indicate links, save files, or neither. You can execute FILES also by SWAPping to "FILES" or CHAINing "FILES."

- *RDOS*
- *DOS*

FLOAD

Fast load FDUMPed files.

Format

FLOAD magtapefile *[filename$_1$ filename$_2$ ...]*

Arguments

magtapefile — The name of a tape file in the form MTn:file, where n is the drive number and file is the tape file number; or the EQUIValent name of a tape drive.

filename — The names of files you want to be selected for loading; you can use asterisk (*) and dash (-) conventions.

Global Switches

/A Load permanent as well as nonpermanent files.

/F Loading will not begin with the first tape volume. Use this when you are loading selected files and they are not on the first tape volume.

/I Ignore any tape read errors.

/K Do not load links.

/L List loaded files in $LPT (overrides /V and /P).

/N Load only links.

/O Delete current file if it exists and replace with *filename* being loaded.

/P Do not load files, but list names on terminal.

/R Load most recent version of the file.

/V Verify loaded files with a listing of names on the terminal.

Licensed Material-Property of Data General Corporation

FLOAD (continued)

- *RDOS*
- *DOS*

Local Switches

mm-dd-yy/A Load only files created on or after this date, where date is in the form mm-dd-yy.

mm-dd-yy/B Load only files created before this date, where date is in the form mm-dd-yy.

filename/N Do not load files matching filename; you can use asterisk (*) and dash (-) conventions.

Why Use It?

Use FLOAD to load FDUMPed files from mag tape; FLOAD is the only way to load FDUMPed files (see FDUMP).

What It Does

FLOAD loads all nonpermanent files into the current directory; or it loads all files if you use the /A global switch. You can either load or list selected files specified in *filename.*

Example

```
!FLOAD/L/R MT0:0 -.SR 2-1-78/A TMP-.-/N )
```

Load from tape file MT0:0 the most recent copies of all files with the .SR extension, created on or after February 1, 1978, that do not begin with TMP, and list their names on the terminal.

FPRINT

Dump a file to the line printer or terminal.

Format

```
FPRINT filename
```

Arguments

filename A file in the current directory.

Global Switches

/B List in byte format.

/D List in decimal.

/H List in hexadecimal.

/L List on $LPT.

/N Don't suppress duplicate lines of output.

/O List in octal.

Local Switches

[O] n/F Print contents starting at octal location n.

Hn/F Print contents starting at hexadecimal location n.

Xn/F Print contents starting at decimal location n.

outputfile/L List to specified outputfile (overrides global /L).

n/O Start at offset n into file.

n/T Last location to be printed is n.

Why Use It?

Use FPRINT to print the contents of a file with a location counter. Contents are printed in octal, decimal, byte, or hexadecimal. The location counter is always printed in octal. The default listing device is the terminal.

Example

```
!FPRINT/H/L MYFILE 200/F 600/T )
```

Print MYFILE in hexadecimal to the lineprinter from location 200 to 600.

Licensed Material-Property of Data General Corporation

- *RDOS*
- *DOS*

FREE

Free a device from exclusive (ASG) use.

Format

FREE filename

Arguments

filename The name of a reserved file assigned to a job by ASG.

Switches

None.

Why Use It?

Use FREE to release a device from exclusive use. FREE frees filename if it has been assigned by ASG (see ASG in this chapter). Only the job that assigned the filename can FREE filename.

Example

```
!FREE $LPT )
```

Free the line printer from exclusive use.

- *RDOS*
- *DOS*

GDIR

Display the current directory name.

Format

GDIR

Switches

None.

Why Use It?

Use GDIR to find out which directory you're in.

What It Does

GDIR returns the name of the current directory. It does not return the name of the system directory (see GSDIR).

Example

```
!GDIR )
```

NBASIC
!

You are currently in directory NBASIC.DR.

Licensed Material-Property of Data General Corporation

- *RDOS*
- *DOS*

GSDIR

Display the current system directory name.

Format

GSDIR

Switches

None.

Why Use It?

Use GSDIR to find out what system directory BASIC is in.

What It Does

GSDIR returns the name of the directory in which BASIC resides.

Example

```
!GSDIR )
DP0
!
```

The BASIC system is in DP0.

- *RDOS*
- *DOS*

GSYS

Display the current system name.

Format

GSYS

Switches

None.

Why Use It?

Use GSYS to find out the name of the operating system.

What It Does

GSYS returns the name of the current operating system.

Example

```
!GSYS )
MYRDOS
!
```

The current operating system is MYRDOS.

Licensed Material-Property of Data General Corporation

- *RDOS*
- *DOS*

GTOD

Display the time and date.

Format

GTOD

Switches

None.

Why Use It?

Use GTOD to display the time as hh:mm:ss and the date as mm/dd/yy.

Example

```
!GTOD )
```

5/22/78 05:35:16

The day is May 22, 1978; the time is 5:35 AM (and 16 seconds).

- *RDOS*
- *DOS*

INIT

Initialize a device.

Format

INIT devicename

Arguments

devicename The name of a device or directory or partition.

Switches

None.

Why Use It?

Use INIT to make a device or disk directory accessible, so you can use the device or access files in the directory.

What It Does

INIT initializes devicename, making it accessible for input/output. Files on multiple file devices (like disks) can't be accessed until the device and any directories containing these files are initialized.

Examples

```
!INIT DP1:USER:MYTOWN )
```

This will initialize DP1 (disk unit 1), the directory USER in DP1, and the subdirectory MYTOWN in directory USER.

```
!INIT MT0 )
```

This will initialize tape drive 0.

Licensed Material-Property of Data General Corporation

- *RDOS*
- *DOS*

LIBRARY

Display names of files in the library directory.

Format

LIBRARY

Switches

None.

Why Use It?

Use LIBRARY to display the filenames of files in the $LIB directory.

What It Does

LIBRARY gives a quick listing of all the nonpermanent files in the Business BASIC library, $LIB, which is open for all to use. The listing will include an indicator L for a link, S for a save file and a blank for neither.

How To Use It

LIBRARY is a BASIC CLI command, but you can also SWAP to or RUN "LIBRARY".

LIBRARY Example

```
!LIBRARY )

S BLACKJACK.SV      S STARTRK.SV      L LINKSTAR        DATA.TB
  FM.TB             L PARAM.TB          CUST.TB         CUST
L PARAM
.
.
.
```


Licensed Material-Property of Data General Corporation

- *RDOS*
- *DOS*

LINK

Link an alternate name to a file.

Format

LINK linkname filename

Arguments

linkname An alias link entry name.

filename A resolution file; can also have a directory specifier.

Switches

None.

Why Use It?

Use LINK to establish an alias link entry in the current directory to a resolution file in the same directory or another directory. LINK to a file in another directory if you need to use that file from your directory. Use UNLINK (not DELETE) to delete a link entry without affecting the resolution file.

What It Does

LINK creates a link entry linkname to use as a reference to filename; filename can be another link entry. The link entry will exist in the current directory. No error message is given if the system can't find the resolution file, filename.

How To Use It

Before you can use a link to access a resolution file, all directories involved in the resolution chain must be initialized. If filename is in another directory, specify the directory in filename. The system gives no error message if it can't find filename, so you should check your link to make sure it works. You can link to a file whose attributes forbid linking (CHATR and CHLAT commands), but you can't use that link.

Examples

```
!LINK BLACKJACK B.SV )
```

A link entry, BLACKJACK, is made in the current directory for B.SV which is also in the current directory.

```
!INIT MYDIR.SR )
!LINK TMP.LK MYDIR:MYFILE.SR )
```

First, we initialize MYDIR (resolution file's directory), then we establish a link entry, TMP.LK, in the current directory for MYFILE.SR in directory, MYDIR.

Licensed Material-Property of Data General Corporation

- *RDOS*
- *DOS*

LIST

List information for files in the current directory.

Format

LIST filename$_1$ *[filename$_2$ ...]*

Arguments

filename The match for a filename entry in your directory; you can use dash (-) and asterisk (*) conventions.

Global Switches

/A List permanent as well as nonpermanent files.

/B Brief listing, give filenames only.

/C List creation time as mm/dd/yy hh:mm.

/E List all file information (overrides /B, /C, /F, /O, and /U).

/F List (in octal) logical address of first block in file (0 if unassigned).

/K Do not list links.

/L List at $LPT (line printer).

/N List links only.

/O List date the file was last opened as mm/dd/yy.

/P Sort list by date of last access.

/Q Quick listing of names only; printed 4 or 8 names across.

/S Sort list alphabetically.

/U List file use count (decimal).

/W Sort list by time of last write.

/X Sort by file size, ascending.

Local Switches

mm-dd-yy/A List only files created on or after this date, where date is in the form mm-dd-yy.

mm-dd-yy/B List only files created before this date, where date is in the form mm-dd-yy.

filename/N Do not list files matching this name, where filename is the name and can contain asterisk (*) and dash (-) conventions.

Why Use It?

Use LIST to list information about directory entries. The information can include file size, resolution attributes, link access attributes, file creation date and time, date last opened, file starting address, and decimal file use count.

What It Does

LIST lists information about one or more files or link entries in the current directory. If you specify *filename$_2$*, LIST only lists the filenames that match *filename$_2$*. Link access attributes, if any, are preceded by a backslash (\). For link entries, LIST lists the name of the link and resolution filenames, the directory specifier (if any) given when the link was created, and an @ sign to indicate that the link was originally made to a file on its directory's parent partition or diskette. An O following the use count indicates the file is currently opened, but may be opened by other users; and an E following the use count indicates the file is exclusively opened and may not be opened by other users. LIST lists resolution or link access attributes:

P Permanent file, can't be deleted or renamed.

S Save file (compiled, core image), or BASIC program file.

W Write protected file, can't be written to.

R Read protected file, can't be read.

A Attribute protected file whose attributes can't be changed (you can't remove the A attribute).

N No resolution entry allowed; links can be made but not used.

& User-defined attribute.

? User-defined attribute.

Licensed Material-Property of Data General Corporation

LIST (continued)

LIST also lists file characteristics:

D File is randomly organized.

C File is contiguously organized.

T File is a partition (RDOS).

Y File is an RDOS or DOS directory.

LIST Example

!LIST/A/E/K -.SR)

List all information for files, but not links, with the .SR extension in the current directory to the terminal. A typical response would be:

MAIN.SR 820 WD/P 01/18/78 14:29 02/25/78 [002164] 0

filename
Size in bytes
Attributes
Date/time last written to
Date last opened
Location of first block
Use Count

MAIN.SR is 820 bytes long, resolution attributes are write protected, and the file is random, has a link attribute of permanent, was last written to at 2:29 PM of February 18, 1978, was last opened February 25, 1978, the first block of the file resides as disk block 2164 (octal), and is currently unopened. The combination of attributes for MAIN.SR would not allow this file to be deleted via a link.

Licensed Material-Property of Data General Corporation

- *RDOS*
- *DOS*

LOAD

Reload dumped files.

Format

LOAD inputfile *[filename$_1$ filename$_2$ ...]*

Arguments

inputfile The name of a given device or directory that contains files previously dumped using DUMP.

filename A match for filename entries in inputfile.

Global Switches

/A Load permanent as well as nonpermanent files.

/B Display a brief listing of loaded files.

/I Ignore any checksum errors.

/K Do not load links.

/L List loaded files to $LPT (overrides /V and /P).

/N Load only links.

/O Delete current file if it exists and replace with file of the same name being loaded.

/P Do not load files, but display names on the terminal.

/R Load most recent version of the file. When a file to be loaded has the same name as a file in the current directory, the system checks both file creation dates. If the existing file is older, it is deleted and replaced by the file awaiting loading. If the existing file is not older, it is retained and the dumped file is not loaded.

/V Verify loaded files by displaying their filenames at the terminal.

Local Switches

mm-dd-yy/A Load only files created on or after this date, where date is in the form mm-dd-yy.

mm-dd-yy/B Load only files created before this date, where date is in the form mm-dd-yy.

filename/N Do not load files matching the name filename; you can use dash (-) and asterisk (*) conventions in filename.

Why Use It?

Use LOAD to load previously DUMPed files. LOAD can't load files dumped by FDUMP (use FLOAD) or by XFER.

What It Does

LOAD loads previously dumped files from inputfile into the current directory. If you omit *filename* and switches, all nonpermanent files are loaded. You can select only files matching *filename* to be loaded, and using switches, you can simply display the filenames in inputfile. DUMP and LOAD do not change a file's attributes or characteristics. Files to be loaded must bear different filenames from files in the current directory (unless you specify the /N, /O, or /R switches).

Example

```
!LOAD/L/R MYDIR.DP -.SR 2-1-78/A TMP-.-/N )
```

Load from the file, MYDIR.DP, the most recent copies of all files with the .SR extension, created on or after February 1, 1978, that do not begin with TMP, and list their names on the line printer.

Licensed Material-Property of Data General Corporation

- *RDOS*
- *DOS*

LSTCOM

Compare two listing files.

Format

LSTCOM listfile$_1$ listfile$_2$ *[outputfile]*

Arguments

listfile	filename of a listing file (character format).
outputfile	filename of the file you want to receive comparisons; if not specified, comparisons go to the terminal or the line printer.

Global Switches

/L Output to $LPT (overrides *outputfile*).

Why Use It?

Use LSTCOM to compare two listing files either created by the LIST command (*not* the CLI LIST command) or by the EDIT utility.

What It Does

LSTCOM compares, line by line, two list files and outputs all lines that differ either to the terminal, to *outputfile*, or to the line printer (/L switch).

Example

!LSTCOM NEWPROG.LS OLDPROG.LS LST)

Suppose NEWPROG.LS contains:

```
0100 INPUT A
0110 FOR I=1 TO A
0120   PRINT I*A
0130 NEXT I
0140 STOP
```

Suppose OLDPROG.LS contains:

```
0100 INPUT A,B
0110 FOR I=1 TO A
0120   PRINT I*A
0125   LET B=B*A
0127   PRINT B
0130 NEXT I
0140 STOP
```

After the LSTCOM command, the output file LST would contain:

```
NEWPROG.LS:  0010 INPUT A
OLDPROG.LS:  0100 INPUT A,B

OLDPROG.LS:  0125 LET B=B*A

OLDPROG.LS:  0127 PRINT B
```


Licensed Material-Property of Data General Corporation

- *RDOS*
- *DOS*

LSTMERGE

Merge two listing files.

Format

LSTMERGE listfile commentedfile *[outputfile]*

Arguments

listfile	Filename of a list file created by the BASIC LIST command.
commentedfile	Filename for a list file with comments (created by the BASIC LIST command and/or the EDIT utility).
outputfile	An optional file to which you can send the results from merging listfile and commentedfile.

Global Switches

/L List output to $LPT (overrides *outputfile*).

Why Use It?

Use LSTMERGE to merge the comments from commentedfile with the BASIC program listing in listfile to produce an updated version of a commented program. Many users use the EDIT utility to create comments for a program listing (comments are preceded by a colon). LSTMERGE reduces the need to EDIT the commentedfile when making small changes to a program.

What It Does

LSTMERGE merges the line in listfile with its matching line number in commentedfile to pick up the comments for that line. All lines in listfile are merged with commentedfile and sent to *outputfile*, or if you do not specify *outputfile*, they remain in commentedfile. Lines in commentedfile which contain comments only will unconditionally be included in the output.

How To Use It

Create your program listing in commentedfile using either the EDIT utility, the BASIC LIST command, or both. When you create an uncommented listing file for the updated program (using LIST), don't replace old lines with new lines using the same line numbers, unless you add new comments to the old line numbers (otherwise you'll get an old comment on a new line). You can delete any comments you don't want by using EDIT on the commentedfile. *Don't* RENUMBER before using LSTMERGE -- you'll lose comments for lines that don't exist in commentedfile.

Examples

```
!LSTMERGE/L PROG.SL PROG.SR )
```

List on $LPT PROG.LS with the comments from PROG.SR.

Licensed Material-Property of Data General Corporation

- *RDOS*
- *DOS*

MDIR

Display the master directory name.

Format

MDIR

Switches

None.

Why Use It?

Use MDIR to find out the name of the master directory.

Example

```
!MDIR )
DP0
```

The current master directory name is DP0.

- *RDOS*
- *DOS*

MOVE

Move files from one directory to another.

Format

MOVE directory *[filename$_1$ filename$_2$ ...]*

Arguments

directory — Filename of the directory you want to receive the files; can be a partition or subdirectory.

filename — Filename for a file in the current directory you want to be used as a match; you can use asterisk (*) and dash (-) conventions. *Filename* can't have a directory or device specifiers as prefixes.

Global Switches

/A Move permanent as well as nonpermanent files.

/C Used with /D. Repeat each filename and wait confirmation that file is to be deleted. Hit carriage return to delete, and any other character to skip.

/D Delete original files once all transfers are complete (i.e., after copying them).

/K Do not move links.

/L List moved files on $LPT (overrides /V).

/N Move only links.

/O Deletes directory: *filename*, if it exists, before moving *filename* to directory.

/P Move files in order of date last opened.

/R Move most recent copy of file. If entry already exists in the destination directory with a more recent creation date, the file is not moved.

/S Move files in alphabetical order.

/V Verify moved files by displaying them at the terminal.

/W Move files in order of time of last write.

Licensed Material-Property of Data General Corporation

MOVE (continued)

- *RDOS*
- *DOS*

Local Switches

mm-dd-yy/A Move only files created on or after this date, where date is in the form mm-dd-yy.

mm-dd-yy/B Move only files created before this date, where date is in the form mm-dd-yy.

filename/N Do not move files matching the name filename; you can use dash (-) and asterisk (*) conventions.

Why Use It?

Use MOVE to copy one or more files from the current directory to the destination directory you specify. You can MOVE all the files in a directory, but you cannot MOVE a directory.

What It Does

MOVE without *filename* or switches copies all nonpermanent entries in the current directory to directory. Only resolution and link entries can be MOVEd; directories and partitions can't be MOVEd. If you specify *filenames*, MOVE searches the current directory for *filenames* and moves them only; you can use the dash (-) and asterisk (*) conventions in *filename*. MOVE does not change the file's characteristics or information in the filename entry.

NOTE: DOS users -- if you copy a file with MOVE from a write-protected diskette, the copy automatically receives the attributes APW, which means that you can never delete or modify the copy.

Example

```
!MOVE/R BKUP -.SR )
```

Move the most recent copies of files with the .SR extension to the directory BKUP.DR.

POP

Exit the BASIC CLI and clear common area.

Format

```
POP
```

Switches

None.

Why Use It?

Use POP to exit the BASIC CLI and return either to the program that SWAPped to CLI or to keyboard mode.

What It Does

POP terminates Business BASIC's CLI program. If you used CHAIN or RUN to execute the BASIC CLI, POP returns you to keyboard mode. If you used SWAP in a BASIC program to execute the BASIC CLI, POP returns you to the BASIC program. The first word of the common area (used to pass CLI commands) is set to <255>; the remainder is set to <0>.

Example

```
!POP )

*

0010 SWAP "CLI"
0020 PRINT "CLI POPPED OUT"
*RUN )
!POP )
CLI POPPED OUT
*
```


Licensed Material-Property of Data General Corporation

- *RDOS*
- *DOS*

PRINT

Print a file on the line printer.

Format

PRINT filename$_1$ *[filename$_2$ ...]*

Arguments

filename The name of a character format (ASCII) file, created by EDIT, the BASIC LIST command, or an XFER/A command; or any text (ASCII) file.

Global Switches

/A Used with /X. Include permanent symbols in cross-reference list.

/D Used with /X. Allow duplicate symbols in cross-reference list.

/H Print a heading at the top of each page.

/O Override print suppression of library subroutines, used with /H or /X. See TABLE later in this chapter.

/X Print a symbol cross-reference list at the end of the listing. See the BASIC CLI command, TABLE

Why Use It?

Use PRINT to copy the contents of ASCII text files to the line printer. PRINT is equivalent to a series of XFER/A filename $LPT commands. To print a binary file, use FPRINT. PRINT will only print an ASCII file. If your file is a source program (with comments) created by the BASIC LIST command (or the EDIT utility), you can also print a cross-reference list of all variables and their corresponding statement numbers using the /X switch, which performs a TABLE. See the BASIC CLI command, TABLE.

Example

```
!PRINT/H/O MAIN.SR )
```

Print the source file MAIN.SR with a heading on each page, allowing subroutines that begin with a REM\comment statement also to be printed.

- *RDOS*
- *DOS*

PRTCOM

Print a BLDCOM documentation file.

Format

PRTCOM filename

Arguments

filename The name of a documentation file produced by BLDCOM.

Global Switches

/F Print each module's comments on a new page.

/L Print comments on $LPT (overrides /O).

Local Switches

filename/O Print comments to the specified output file, filename.

"symbols"/S If specified, only those symbols defined in symbols will be printed; if not, all symbols will be printed. See "What It Does" for the symbols.

Why Use It?

Use PRTCOM to print a documentation file built by BLDCOM for selective printing of comments with descriptions.

What It Does

When you have these symbols as the second character of a comment symbol, (e.g., :!, ;!, C!, etc.) they mark the beginning of specific sections of comments and the following headings are printed:

Symbol	Heading
!	(Module name and optional short description.)
>	*CALLING SEQUENCE*
#	*DESCRIPTION*
$	*EXTERNAL STORAGE*
&	*EXTERNAL ROUTINES*
%	*EXTERNAL PROGRAMS*
@	*FILES USED*

Licensed Material-Property of Data General Corporation

PRTCOM (continued)

You can specify one or more of these symbols in symbols/S to print only selected portions of the documentation file, or you can print all portions by not specifying symbols. You have to put these special characters in your comment file as the second character after the comment character (: for BASIC, ; for assembly language comments, and C for FORTRAN comments). Use the EDIT utility to put these comments and symbols in your source or listing file.

Example

```
!BLDCOM VAL.SL DIV.SL )
DOCUMENTATION FILE: TMP )
```

Build a documentation file of the comments of subroutines VAL.SL and DIV.SL.

```
!PRTCOM TMP )
```

Print all the comments for the symbols.

- *RDOS*
- *DOS*

QUIT

Exit the BASIC CLI (retains common area).

Format

QUIT

Switches

None.

Why Use It?

Use QUIT to terminate the BASIC CLI yet leave the common area alone for you to examine when debugging your own CLI commands. You can then use GETCM.SL (subroutine in Chapter 7) or BLOCK READ to read your common area.

What It Does

QUIT performs a POP without clearing the common area (see POP in this chapter).

Examples

```
.
.
.
0100 SWAP "CLI"
0110 BLOCK READ X$  :CLI quits, read common.
.
.
.
*RUN )
.
.
!QUIT )
.
.
```


Licensed Material-Property of Data General Corporation

- *RDOS*
- *DOS*

RELEASE

Release a device or directory.

Format

RELEASE { unit / directory }

Arguments

unit A disk, tape drive, or other suitable device.

directory The name of a subdirectory or partition.

Switches

None.

Why Use It?

Use RELEASE to release any disk or tape drive that has been INITed. You must RELEASE a disk before you remove it from the disk unit. You can also RELEASE a directory to prevent further I/O access to it.

What It Does

RELEASE automatically rewinds a mag tape. After release, a directory or device is closed to access until you initialize it with an INIT or DIR command. Releasing a directory also releases all subordinate directories.

Examples

```
!RELEASE DP1 )
```

Disk unit DP1 is released, you can remove the disk pack from drive 1.

```
!RELEASE MT0 )
```

Mag tape unit 0 is released, and the tape is rewound.

- *RDOS*
- *DOS*

RENAME

Rename files.

Format

RENAME oldname$_1$ newname$_1$ *[oldname$_2$ newname$_2$ ...]*

Arguments

oldname A filename for an existing file in the current directory.

newname A new filename for the file.

Switches

None.

Why Use It?

Use RENAME to rename any nonpermanent file. Use EQUIV to rename a device. Use the BASIC REPLACE command to update a save file. Do not RENAME a utility or subroutine unless you intend to have everyone use the new name.

What It Does

RENAME changes oldname to newname if the file does not have the P (permanent) attribute.

Example

```
!RENAME TMP.SR MYFILE.SR )
```

Renames file TMP.SR to MYFILE.SR in the current directory.

Licensed Material-Property of Data General Corporation

- *RDOS*
- *DOS*

SDIR

Set the system directory.

Format

SDIR directory

Arguments

directory The name of a directory without the .DR extension.

Switches

None.

Why Use It?

Use SDIR to change the current BASIC system directory to another directory.

What It Does

SDIR changes the BASIC system directory to directory.DR, initializing directory.DR if necessary.

Example

```
!SDIR BBASIC )
```

Changes the current system directory to BBASIC.DR.

- *RDOS*
- *DOS*

SLINE

Select a line (terminal) and attach.

Format

SLINE port/L *[width/W]*

Arguments

See local switches.

Global Switches

/P Page up (form feed) the attached terminal.

Local Switches

port/L Job should attach to the line number port for the port (terminal) you want. This switch is required.

width/W Optional, set the destination port's page width to *width*.

Why Use It?

SLINE allows the current job to attach to port number port. You can optionally set the page width for the destination port, and output a form feed to the port. The specified port must not be in use by another job. Find out what ports are available by running by the PORTS utility program or the STAT CLI command.

SLINE Example

```
!START CLI/N "SLINE/P 10/L 132/W;TYPE/H MYPROG.SR;BYE" )
```

Start a job (see START in this chapter) running attached to the print only terminal on port 10 beginning with a page-up and page width set to 132 columns.

Licensed Material-Property of Data General Corporation

SPDIS

Disable device spooling.

Format

SPDIS device$_1$, *[device$_2$ ...]*

Arguments

device Any device defined as *spoolable* (defined below).

Switches

None.

Why Use It?

Use SPDIS to disable spooling on a device. RDOS automatically spools data sent to any user device defined as *spoolable,* and to all the following devices: $DP0, $LPT, $LPT1, $PLT, $PLT1, $PTP, $PTP1, $TTP, $TTP1, and the teletypewriter versions of $TTO and $TTO1.

What It Does

During a spool operation, RDOS queues data sent to output devices in disk buffers; this frees the CPU for further processing while the devices receive the queued data. If a spool operation requires more disk space than is available, the system itself will issue the equivalent of an SPDIS command.

After you type the SPDIS command, the system will stop spooling data to the disabled device's disk buffer. Output will continue until all data waiting in the buffer has been processed.

Example

```
!SPDIS $LPT )
```

This command prevents data output to the line printer from being spooled. To reinstitute spooling, you must issue the command SPEBL $LPT. If output is currently being spooled to the line printer, spooling will stop after the current spool is completed.

• *RDOS*
DOS

SPEBL

Enable device spooling.

Format

SPEBL device$_1$ *[device$_2$ ...]*

Arguments

device Any device defined as *spoolable* (defined below).

Switches

None.

Why Use It?

Use SPEBL to re-enable spooling on a device for which spooling has been disabled. The device may be any user device defined as *spoolable,* or any of the following: $DP0, $LPT, $LPT1, $PLT, $PLT1, $PTP, $PTP1, $TTP, $TTP1, and the teletypewriter versions of $TTO and $TTO1.

RDOS does not automatically spool to the plotter ($PLT, $PLT1), thus you must explicitly enable spooling to these devices.

Example

```
!SPEBL $LPT )
```

Re-enable spooling to the line printer.

Licensed Material-Property of Data General Corporation

• *RDOS*
DOS

SPKILL

Delete the spool queue.

Format

SPKILL device$_1$ *[device$_2$ ...]*

Arguments

device Any device defined as *spoolable* (defined below).

Switches

None.

Why Use It?

Use SPKILL to stop a spool operation by deleting the spool files enqueued to device. The device may be any user devices defined as *spoolable* or any of the following: $DPO, $LPT, $LPT1, $PLT, $PLT1, $PTP, $PTP1, $TTP, $TTP1, or the teletypewriter versions of $TTO and $TTO1.

What It Does

After you kill spooling on a device, data on the output spool is lost. Spooling will resume to the device on the next command which sends data to the device; e.g., a BASIC CLI PRINT command.

Example

```
SPKILL $LPT )
```

This command stops the spooling of data to the first line printer.

• *RDOS*
• *DOS*

START

Start a de-attached job.

Format

START *[prog/N] [accnt/A] [passw/P] [dir/D] ["input ;"] ["input ;"...]*

Arguments

input A message, in quotes, you supply as input to the job.

Global Switches

/V Verify start-up by displaying job dialog as it is transmitted.

Local Switches

prog/N Name of program (to be run as de-attached job) is *prog.*

accnt/A De-attached job logs on with *accnt* as account ID.

passw/P Password is *passw* for specified account, required with */A.*

dir/D De-attached job runs from directory *dir.*

Why Use It?

Use START to start a program running as a de-attached job where you can send messages *(input)* as input to the de-attached program.

What It Does

START starts the program *prog* running as a de-attached job. When you specify *accnt/A* and *passw/P,* the job is logged on accordingly; otherwise the de-attached job's account will be the current account executing START. If you specify *dir/D,* the de-attached job will run in directory *dir;* otherwise, the job will run from the current directory. Optionally there may be one or more input messages *(input),* supplied as input to the de-attached job. If no *prog* is specified, the job is left in keyboard mode and any *input* is passed to the BASIC interpreter.

Licensed Material-Property of Data General Corporation

WARNING: If the started job does not issue INPUT requests for the supplied *input*, control will not return to your terminal. STARTed jobs will normally run at a lower priority than the job executing the START.

Example

```
!START CLI/N TDATA/D "FDUMP/A/L MT0:0;BYE" )
```

Start a fast dump of the TDATA directory. You can proceed with your work while the dump is happening without having to log back on. Note that BYE is used as the last command so the job will be forced off the system when it finishes.

STAT

Display the status of all jobs.

Format

```
STAT
```

Global Switches

/A Include the user account number.

/D Include the user directory specifier.

/I Include the I/O usage (number of system calls).

/L Include the job log on time.

/N Include the program name.

/P Include the program push level.

/Q Include the amount of CPU time used.

/R Repeat the STAT output until interrupted.

/S Include the program size in bytes (program and data).

/T Include the terminal line number.

/X Include the job task priority.

Why Use It?

Use STAT to find out what's happening on your system.

What It Does

STAT executed as a CLI command without switches displays on your terminal -- the active job numbers, their I/O status, their locations, and their run status (output sections a,b,c, and d in the example below). STAT with a switch includes the output specified by the switch. You can also RUN "STAT" or SWAP or CHAIN "STAT" (no switches allowed), and STAT will display all the information.

Licensed Material-Property of Data General Corporation

STAT (continued)

STAT Example

```
!STAT/A/D/I/L/N/P/Q/S/T/X )

00      C  R  13  DOC    STAT      AATLB6  03  1   3428   55.7   2507  14:47
01  I   D  R  13  DOC    EDIT      ABDGC6  02  0   8914   88.2   5025  15:45
02  I   D  C  13  OMI    SCRATCH   ABEAS6  01  0   10228  113.9  1462  15:49

↑   ↑   ↑  ↑  ↑  ↑      ↑         ↑       ↑   ↑   ↑      ↑      ↑     ↑
a.  b.  c. d. e. f.     g.        h.      i.  j.  k.     l.     m.    n.
```

a. The job number.

b. Indicator showing the current I/O status where:

I	waiting on input
O	waiting on output
S	waiting on system call
D	waiting on DELAY
L	waiting on LOCK
blank	I/O satisfied

c. Indicator showing the current location of the program where:

A	program in core; main partition
M	program in extended memory (hyperspace); not in user partition
D	program on disk (swapped out)

d. Indicator showing what the current program is doing:

R	program is running
C	program is in compile state
B	program is both running and compiling

e. The current program task priority.

f. The job's directory specifier.

g. The name of the current program.

h. Account code job used at log on.

i. The job's terminal line number; -1 if de-attached.

j. The program push level for this job.

k. The program's size in bytes (statements and data).

l. The CPU time in seconds used since log on.

m. The number of system calls made since log on (I/O usage).

n. The time of log on for this job.

Licensed Material-Property of Data General Corporation

- *RDOS*
- *DOS*

TABLE

Print a program cross-reference.

Format

TABLE filename$_1$ *[filename$_2$ ...]*

Arguments

filename The name of a listing file (of a program) in character format (ASCII) created by a BASIC LIST command or the EDIT utility.

Global Switches

/A Include permanent symbol in cross-reference list (LET, IF, etc.)

/D Allow duplicate entries in cross-reference list.

/H Prints a heading at the top of each page.

/L Print output on $LPT (line printer).

/O Override print suppression of library subroutines.

/P Include a program listing before cross-reference list.

Local Switches

outputfile/L Append listing to outputfile (overrides global /L).

Why Use It?

TABLE prints a cross-reference table of all variables in the program in filename and the statement numbers where they reside.

What It Does

TABLE also prints a listing of the program like the CLI PRINT command. When TABLE encounters a statement in the form:

```
REM comment
```

It will not print the lines following it until another REM statement in the same form is encountered. This is used to suppress printing of the library subroutines (subroutines in Chapter 7). You can override this suppression with the /O switch.

TABLE Example

```
!TABLE/H/P/L TEST.LS SORT.SR LSTFILE/L MYFILE.LS )
```

Prints TEST.LS and SORT.SR with cross-reference list and page headings, on the line printer, and list MYFILE.LS to the output file LSTFILE.

Licensed Material-Property of Data General Corporation

- *RDOS*
- *DOS*

TCOPY

Copy from tape to tape.

Format

TCOPY inputfile *[outputfile] [number/N]*

Arguments

inputfile The name of a mag tape file or intermediate disk file you want input to TCOPY.

outputfile The name of a mag tape file or intermediate disk file you want to receive output.

Global Switches

/E Extended verification, reports number of records found for each record size found.

/I Ignore any tape read errors.

/L Verify copied files to $LPT (overrides /V and /E).

/R Read input file only. Allows /E or /V without actual copying.

/V Verify copied files on the terminal.

Local Switches

filename/L List verification information to filename.

number/N Number of consecutive files to be copied.

Why Use It?

Use TCOPY to copy one or more files from a mag tape drive to another mag tape drive, or copy to an intermediate disk file if you have only one tape drive.

What It Does

By default, all files on a tape (or disk file) beginning with the one specified as inputfile are copied through the last file on tape. You can optionally specify a specific number of files from the tape (or disk file) by specifying number/N. TCOPY also prints a verification like the one in our example. The files on the tape (or disk file) may have records of varying length up to a maximum record size of 8192 bytes. Records of an odd number of bytes can't be duplicated properly due to hardware restrictions.

Examples

```
!TCOPY MT0:3 MT1:0 1/N )
```

This will copy file 3 on tape unit 0 (and only file 3) to file 0 of tape unit 1.

```
!TCOPY/E/R MT0:0 )
```

Read all files on tape unit 0 beginning with file 0 and report the number of records of each size found.

Licensed Material-Property of Data General Corporation

- *RDOS*
- *DOS*

TFER

Copy a file between tape and disk.

Format

TFER inputfile outputfile recsize/R *[blocksize/B]*

Arguments

inputfile and outputfile — Tape or disk files. Tape files are recognized by the colon (:), so you can LINK or EQUIV a tape drive, but *not* a tape file. Disk files must be in the current directory; no directory specifier is allowed.

Global Switches

/A Convert EBCDIC code to ASCII code.

/E Convert ASCII to EBCDIC.

/V Variable length records (for ASCII text files).

Local Switches

blocksize/B — Number of records in block; optional, default is 1.

recsize/R — Record size in bytes; required. To read RDOS format tapes, set recsize equal to 510.

Why Use It?

You can use TFER to copy a tape to disk or a disk to tape allowing ASCII/EBCDIC conversions. You can access tape blocks of up to 8192 bytes, and you can have variable length ASCII records. TFER *does not* produce an RDOS dump format tape (all tape I/O is via MTDIO -- see MTDIO in Chapter 3).

What It Does

TFER copies blocks from tape to disk or from disk to tape. A tape file is recognized by the colon (:) separating the drive from the file, so you can use a LINK or EQUIV for the drive, but not for the drive and the file. If you use a directory specifier, TFER will interpret it (because of the colon) as a tape drive. If you specify (/V) variable length records, TFER will pad with spaces to the specified record size or truncate characters if it is longer than the specified record size. If you want to change fixed length records to variable, TFER truncates trailing spaces.

How To Use It

You must specify recsize/R. If you don't specify blocksize/B, TFER will assume you want 1 record per block. To read an RDOS format tape, you must specify 510 for recsize. You can't write an RDOS format tape with TFER; you must use XFER.

Examples

```
!TFER/A/V MT0:0 SOURCE.SR 80/R 20/B )
```

Copy EBCDIC tape file with 80-byte records (20 records per block) to disk file SOURCE.SR converting to ASCII and dropping trailing spaces.

```
!TFER/E/V MYFILE.SR MT0:0 80/R )
```

Copy ASCII source file on disk in current directory to tape MT0:0, with 80-byte records (1 record per block), converted to EBCDIC and padded with spaces.

Licensed Material-Property of Data General Corporation

TPRINT

Print the tuning report.

Format

TPRINT

Switches

None.

Why Use It?

The information given by TPRINT is useful for optimizing the structure of subsequent generations of RDOS and Business BASIC systems.

What It Does

TPRINT prints information in the RDOS tuning file and some Business BASIC information. The RDOS tuning file is in the directory which holds the current RDOS system, if tuning was specified during RDOS system generation. It contains information on system stacks, cells, buffers, and overlays. In addition, Business BASIC has lock buffers, BASIC overlays, and the overlay area. For each of these, TPRINT displays the number in the system, the number of requests, the number of faults, and the percentage of faults. A fault is a request for the item which had to be delayed because the item was not immediately available. This command is useless unless RDOS turns on system tuning (see *RDOS/DOS CLI User's Manual.*

TPRINT Example

```
!TPRINT )
```

	NUMBER	*REQUESTS*	*FAULTS*	*PERCENT*
STACKS	*4*	*1107318*	*154984*	*13%*
CELLS	*12*	*982746*	*0*	*0%*
BUFFERS	*11*	*1040507*	*30655*	*2%*
OVERLAYS	*54*	*970043*	*26445*	*2%*
LOCK BUFS.	*5*	*0*	*0*	*0%*
BASIC OLYS	*8*	*7737*	*1358*	*17%*
OLY AREA	*1*	*7737*	*4*	*0%*

This might suggest that you need more stacks to improve system throughput, and you need less cells to gain more core space (i.e., make your system smaller).

Licensed Material-Property of Data General Corporation

- *RDOS*
- *DOS*

TYPE

Display a file on your terminal.

Format

TYPE filename$_1$ *[filename$_2$ ...]*

Arguments

filename The name of a text (ASCII) file; can be a program listing file created by the BASIC LIST command or the EDIT utility.

Global Switches

/A Used with /X. Include permanent symbols in cross-reference list.

/D Used with /X. Allow duplicate symbols in cross-reference list.

/H Print a heading at the top of each page (screen page).

/O Override print suppression of library subroutines, used with /H or /X (see TABLE in this chapter).

/X Print a symbol cross-reference list at the end of the listing (see TABLE in this chapter).

Why Use It?

Use TYPE to display a character format (ASCII) file on your terminal. To type a binary file, use FPRINT. Use the global switches if you also want to display a cross-reference table for listing files (program source or listing files).

Example

```
!TYPE MYFILE.SR YOURFILE.SR )
```

The contents of MYFILE.SR and YOURFILE.SR will appear on the terminal.

- *RDOS*
- *DOS*

UNLINK

Remove a link entry.

Format

UNLINK linkname$_1$ *[linkname$_2$ ...]*

Arguments

linkname The name of a link entry established by a LINK command (see LINK in this chapter). You can use dash (-) and asterisk (*) conventions.

Global Switches

/C Confirm each removal. The system repeats each link name on the terminal and waits for you to confirm the deletion by typing a carriage return. To prevent deletion or unlinking, hit any key other than carriage return.

/L List deleted entries on $LPT (overrides /V).

/V Verify link entries deleted on the terminal.

Local Switches

filename/N Do not delete links matching the name filename. You can use dash (-) and asterisk (*) conventions in filename.

Why Use It?

Use UNLINK to delete link entries created by the LINK command.

What It Does

UNLINK deletes entries from the current directory without affecting the resolution file to which the link points.

Example

```
!UNLINK/V MYFILE TMP.LK TMP-.SR )
```

Delete and list on the terminal all link entries that begin with TMP and have the .SR extension, and the links MYFILE and TMP.LK.

Licensed Material-Property of Data General Corporation

VFU

Edit a format control file for a data channel line printer.

Format

VFU *[switches]* vfu-filename

Arguments

vfu-filename The name of VFU file (with a .VF extension) you are creating or accessing (see "How To Use It").

Global Switches

/C Create a new VFU file.

/E Edit an existing VFU file.

/F Line printer has full character set (upper and lowercases.) Used with /X.

/L List VFU file to the line printer.

/N Line printer has non-DMA controller. Used with /X.

/P Print alignment file after loading VFU (default file: ALIGNVFU).

/T Display and/or alter tab settings DMA printer only).

/S Direct global printer action (/L and /X) to the secondary line printer, $LP2 ($LP2 should be linked to $LPT1).

/V Verify channel and tab settings to the console.

/X Transfer the VFU file to the printer's VFU unit.

Local Switches

/L Listing file to receive VFU file.

/P Alignment pattern file.

Each switch is described in "How To Use It."

Why Use It?

The VFU utility allows you to create and edit format control files for data channel line printers; it can also load format control files into the printer's memory.

You can specify three settings for the printer forms in each vfu-filename: tab stops, form size in lines, and multiple line-number/channel-number pairs.

What It Does

VFU has four main functions:

1. It creates files (always appending the .VF extension to your specified filename).
2. It displays files.
3. It edits existing .VF files.
4. It loads an existing .VF file into the printer's memory.

It can also enable and disable access to the printer's memory by user programs.

How To Use It

You specify each VFU function with a global switch. You must specify an enable or disable command alone; otherwise, you can enter multiple commands with multiple switches. In a multiple command line, the CREATE and EDIT commands are processed first, followed by load commands, then display commands. You need not enter the .VF extension to access VFU files.

Create a .VF File (VFU/C)

When you type VFU/C filename, the VFU program announces itself, then displays the new filename, and asks:

1. TAB CONTROL:
   WANT STANDARD TABS (EVERY 8 COLUMNS)?
   ENTER Y/N ==>

   If you want standard tab stops (at column 0, 7, 15,...127), type Y; VFU then skips to question 3. To set your own tabs, answer N; VFU then asks:

2. ENTER COLUMN NO. (1-132) OR CR ==>

   Enter each column number at which you want a tab stop in this file; press) after typing each number. VFU will repeat this question until you type only); it then asks:

3. VFU CONTROL:
   WANT STANDARD (11 INCH)?
   ENTER Y/N ==>

Licensed Material-Property of Data General Corporation

VFU (continued)

The answers you give to questions 3. and 4. cannot be changed in this file. A *standard* form is 11 inches (66 lines) long, has channel one set for line 1, and channel 12 set for line 63. If you want standard control for the forms which you will print using this file, type Y; VFU then skips to question 5. To specify a different form length, type N; VFU then asks:

4. ENTER FORM SIZE IN LINES (1-143) = = >

   Enter the number of vertical lines in the forms which you will print using this file.

   Next, VFU asks:

5. ENTER LINE NUMBER OF CR = = >

   Specify a vertical line in which you want to set a channel *hole*, then type) VFU asks:

6. ENTER CHANNEL NO. (1-12) = = >

   Specify the channel number which you want to associate with the line number you gave in the last question. VFU then asks question 5 and 6 again; it continues asking these questions until you type) in response to 5.

VFU now creates vfu-filename.VF and returns to the CLI.

Display a File

(VFU/L for first line printer, VFU/V for terminal)

Global Switches

/S Display settings on the second line printer (e.g., VFU/L/S).

Local Switches

filename/L Send settings to filename (overrides global /L).

/V Display vfu-filename's settings on the terminal.

VFU displays the tab-stop and VFU channel settings or sends them to the specified file. It shows line-number/channel settings in the form 1-c, where 1 is the line number of a *hole* and c is the channel number of a *hole*. If you default creation question 3, thus specifying 11-inch forms for a file, VFU channels would be shown as:

1-1 63-12

Edit a File (VFU/E)

If you specify a display option with /E (i.e., VFU/V/E), the vfu-filename will be displayed. VFU then asks about tab control:

1. TAB CONTROL:
   ENTER COLUMN NO. (1-132) OR CR = = >

   Specify a column number where you wish to set a new tab or clear an old tab, and type). Then, VFU asks:

2. ENTER SET(S) OR CLEAR(C) = = >

   To set a tab at the column number specified in step 1., type S; to clear a tab at this column, type C. VFU now asks question 1 again; it continues this loop until you type) in response to 1.

   Now, VFU asks vertical format questions:

3. VFU CONTROL:
   ENTER LINE NUMBER OR CR = = >

   Respond with the line number of a channel *hole* which you wish to set or clear, and). VFU then asks about the channel number associated with this line number:

4. ENTER CHANNEL NO. (1-12) = = >

   Enter the channel number which you want to associate with the line number (for SET) or which is already associated with the line number (for CLEAR). Now, VFU asks:

5. ENTER SET(S) OR CLEAR(C) = = >

   To set a new channel *hole*, type S; to clear an existing *hole*, type C. VFU then repeats questions 3, 4, and 5 until you respond with) to 3.

VFU now updates the existing vfu-filename with the new setting, and displays the new settings on the display file you specified (with /V, etc.).

Load a File into the Printer's Memory (VFU or VFU/X for $LPT, VFU/S for $LPT1)

For the first data channel line printer, type VFU vfu-filename; for the second DCH printer, type VFU/S vfu-filename. VFU then displays a prompt message. When you strike a key and the printer is ready, VFU kills spooling and XFERs the vfu file into the printer's memory. You can then print files on the forms which need the format control contained in the vfu file.

Licensed Material-Property of Data General Corporation

VFU (continued)

Examples

```
!VFU/C PAYROLL1 )
DATA CHANNEL LINE PRINTER FORMAT
CONTROL PROGRAM
CREATING PAYROLL1.VF
TAB CONTROL:
WANT STANDARD TABS (EVERY 8 COLUMNS)?
ENTER Y/N = = > N
ENTER COLUMN NO. (1-132) OR CR 3 )
ENTER COLUMN NO. (1-132) OR CR 9 )
ENTER COLUMN NO. (1-132) OR CR 16 )
ENTER COLUMN NO. (1-132) OR CR 28 )
ENTER COLUMN NO. (1-132) OR CR 50 )
ENTER COLUMN NO. (1-132) OR CR )
VFU CONTROL:
WANT STANDARD (11 INCH)?
ENTER Y/N = = > N
ENTER FORM SIZE IN LINES (1-143) = = > 44 )
ENTER LINE NUMBER OR CR = = > 1 )
ENTER CHANNEL NO. (1-12) = = > 1 )
ENTER LINE NUMBER OR CR = = > 4 )
ENTER CHANNEL NO. (1-12) = = > 2 )
ENTER LINE NUMBER OR CR = = > 9 )
ENTER CHANNEL NO. (1-12) = = > 3 )
ENTER LINE NUMBER OR CR = = > 41 )
ENTER CHANNEL NO. (1-12) = = > 12 )
ENTER LINE NUMBER OR CR = = > )

!
```

This sequence created VFU file PAYROLL1.VF.

The next sequence shows a combined display/edit and loading of this file.

```
!VFU/V/E PAYROLL1 )
DATA CHANNEL LINE PRINTER FORMAT
CONTROL PROGRAM
PAYROLL1.VF  06/15/77 14:22:16
TAB STOPS:
3, 9, 16, 28, 50
VFU CHANNELS:
1-1, 4-2, 9-3, 41-12

EDITING PAYROLL1.VF
TAB CONTROL:
ENTER COLUMN NO. (0-132) OR CR = = > 8 )
ENTER SET(S) OR CLEAR(C) = = > S )
ENTER COLUMN NO. (0-132) OR CR = = > 9 )
ENTER SET(S) OR CLEAR(C) = = > C )
ENTER COLUMN NO. (0-132) OR CR = = > )
VFU CONTROL:
ENTER LINE NUMBER OR CR = = > 14 )
ENTER CHANNEL NO. (1-2) = = > 4 )
ENTER LINE NUMBER OR CR = = > )
PAYROLL1.VF  06/15/77 14:28:25
TAB STOPS:
3, 8, 16, 28, 50
VFU CHANNELS:
1-1, 4-2, 9-3, 14-4, 41-12
!
!VFU PAYROLL1 )

DATA CHANNEL LINE PRINTER FORMAT
CONTROL PROGRAM
PREPARE TO LOAD PAYROLL1.VF
WAIT UNTIL OUTPUT TO THE PRINTER HAS
COMPLETED. MAKE SURE PRINTER IS READY
AND ON-LINE.
STRIKE ANY KEY WHEN READY.
```

Someone now strikes a key.

Licensed Material-Property of Data General Corporation

- *RDOS*
- *DOS*

XFER

Copy one file to another file.

Format

XFER sourcefile destinationfile

Arguments

sourcefile and destinationfile — The names of any device or disk files. Each can include directory specifiers, but neither can be a directory.

Global Switches

/A Perform an ASCII transfer. Transfer the file line by line, taking appropriate read/write action, such as inserting line feeds after carriage returns, etc.

/B Append sourcefile to the end of destinationfile; required if you are copying to an existing disk or tape file.

Local Switches

destinationfile/C Organize the destination file contiguously (both files must be disk files).

destinationfile/R Organize the destination file randomly.

destinationfile/N Do not create destinationfile; it must already exist.

Why Use It?

Use XFER to copy a file in precisely the same format to another file. You can change the organization of a file by using the local switches with destinationfile. Use the /A global switch to perform ASCII (character) transfers.

What It Does

XFER copies sourcefile to destinationfile. When you XFER a file to a disk file, XFER creates destinationfile unless you specify the /N local switch. When you XFER a file to an existing disk or tape file, you must use the /B global switch. The old filename, attributes, creation date, etc. are not copied.

In RDOS, if you omit switches, XFER organizes the destination file *sequentially*. To organize it randomly, you must use the /R local switch. To organize it contiguously, both files must be disk files, and you need to use the /C local switch.

In DOS, if you omit switches, XFER organizes the destination file randomly. In either system, if you XFER a save file, be sure to use CHATR to add an S attribute to the destination file.

Examples

```
!XFER/B SUB.SR MAIN.SR )
```

The file SUB.SR is appended to MAIN.SR.

```
!XFER/A MAIN.SR LSTFILE )
```

The file MAIN.SR is copied in ASCII format to the file LSTFILE.

```
!XFER/A/B TEXT MT0:0 )
```

Copy the file TEXT in ASCII format to the tape file 0 on mag tape drive 0.

End of Chapter

Licensed Material-Property of Data General Corporation

Chapter 9
INFOS Interface

The Business BASIC system optionally provides extensive interface to INFOS® files. This capability allows you to use Indexed Sequential Access Method (ISAM) or Data Base Access Method (DBAM) for INFOS files created on an RDOS INFOS system. The requirements for using this interface are the same as those for RDOS INFOS. You must read the *INFOS System User's Manual (RDOS)* and be familiar with INFOS files and access methods before you try the interface.

INFOS allows you to use a file system with which you can maintain large databases in multiuser environments. The interface supports the INFOS system as well as the standard features of Business BASIC. In addition to the advanced ISAM techniques available with this interface, you can also use DBAM to support database inversion, space management, multiple level indexing, and multiple-user file access.

The ISAM interface allows you to have variable-length records and keys, and permits keyed and relative processing. You can position to a place within the database based on a key, and you can process data in descending or ascending sequence.

DBAM gives the system database orientation. Database inversion lets you create multiple index files that share the same database.

You can store partial records in the DBAM index so that frequently-used data can be found without database access. Extended key features include key compression, duplicate keys, generic, and approximate keys. With multiple level indexes you can define hierarchical data structures and complete data cross-referencing. The hierarchical distribution feature lets application programs distribute database and index records on devices of different speeds based on expected activity levels. You can dynamically delete DBAM records and index entries.

Included in the Business BASIC/INFOS interface are facilities you can use to utilize the INFOS record lock/unlock features. Files maintained by the Business BASIC/INFOS interface are compatible with INFOS ISAM and DBAM files created and accessed by COBOL, RPG, or FORTRAN.

General Format

All Business BASIC INFOS statements have a similar format, although the arguments vary for each function. Throughout this chapter, we use specific variable names, but you can use any variable names you choose. Here is the general format:

NFOS function E, *[C, B$, L]* arguments

where function is one of the following INFOS statement functions.

INFOS Statement Functions

E is the error return variable which is set to zero or to the INFOS I/O error code if an error occurs. You can omit E and the following comma, in which case any error detected causes an I/O error to occur. (You can trap I/O errors with the ON ERR statement.)

C is the BASIC channel number of an INFOS file (0-15).

B$ is a buffer string in which the necessary INFOS packet(s) are built. In some cases, you may wish to set some parameters in B$ before the call, or to examine certain parameters after the call. You must dimension B$ to at least 132 bytes for OPEN, and at least 42+10* < number of keys > for all other functions except RETDF and DEFSI, for which it must be at least 52+10* < number of keys >. If you use the same buffer string for all functions, you should fill it with nulls after an OPEN function and after any function which modifies the buffer. The description of each function lists additional parameters that you can modify and the parameters you may wish to examine. All functions (except OPEN) return PFPEF and PFVER.

L is used only for certain functions and you must describe it with that function. If you omit L, omit the preceding comma.

Licensed Material-Property of Data General Corporation

Many functions require you to include a positioning specification posspec which gives the keyed and relative positioning desired. The posspec is defined as follows:

M;K1$,T1,O1;K2$,T2,02;K3$,T3,03,. . . .Kn$,Tn,On

Where:

M is the relative motion specification you want to apply. If you make M negative, then you will not return to the current position after the function is performed. Otherwise, you will return to the current position after performing a function. You may omit M, in which case zero is the default. Values you can use for M are:

0 No relative motion
1 Forward motion
2 Backward motion
3 Downward motion
4 Down and forward motion
5 Up and forward motion
6 Up and backward motion
7 Upward motion
8 Static motion
9 No relative motion

Kn$ is a key string for keyed accessing. You cannot make Kn$ a null string.

Tn is the key type.

Values for Tn are:

1 Approximate key
2 Generic key
4 Duplicate key

On is the occurrence number of a duplicate key, or the returned occurrence number you assigned in a WRITE statement.

On and the preceding comma may be omitted. If you omit On, then you may omit Tn and the preceding comma. If you do not need accessing, you may omit Kn$ and the preceding semicolon. You can make n any number from 0 to the maximum number of levels you specified when you created the file.

Close an INFOS File (CLOSE)

Format:

NFOS CLOSE E, *[C,B$]*

This statement closes an INFOS file which was opened on channel C. Note that NEW and CLOSE close all open files including INFOS files.

Define a Subindex (DFSI)

Format:

NFOS DFSI E, *[C,B$,L]* ,KL,PL,NS;posspec

Where:

L is the lock/unlock word.

Values for *L* are:

0 No locking or unlocking
2 Lock the index entry
-2 Unlock the index entry

KL is the maximum length of a key in this new index.

PL is the length of the partial records for this index. If you omit PL, zero is assumed and you must also omit the preceding comma and NS.

NS is the minimum node size for this index. If you omit NS, a size of 506 is assumed and you should also omit the preceding comma.

This statement defines a subindex for the index entry you indicated; and uses SDRMF and SDIFL.

Delete an INFOS Record (DELETE)

Format:

NFOS DELETE E, *[C,B$,L];* posspec

Where:

L is the local/global flag. If you omit *L* a value of 0 is assumed.

Values for *L* are:

0 Physically delete the database record and the index entry
1 Logically delete the database record
2 Logically delete the index entry
3 Logically delete both the index entry and the data record

This function physically deletes a record and the index entry, or simply marks them as deleted. A physical delete of the database record does not occur if any other index entries refer to that record. You can reverse a logical delete with the REINS statement. You can only perform a physical delete with keyed access.

Licensed Material-Property of Data General Corporation

Delete a Subindex (DELSI)

Format:

NFOS DELSI E, *[C,B$];* posspec

The subindex is unlinked from the indicated index entry. If no other index entry refers to the subindex, it is physically deleted.

Link to an Existing Subindex (LNKSI)

Format:

NFOS LNKSI E, *[C,B$,L];* $posspec_1$ TO $posspec_2$

Where:

L is the lock/unlock flag.

Values for *L* are:

0 No locking or unlocking
2 Lock both index entries
-2 Unlock both index entries

The subindex at posspec is also linked to the index entry at posspec. This statement returns SRLLD.

Open an INFOS File (OPEN)

Format:

NFOS OPEN E, *[C,B$],* filename

This statement opens the INFOS index file filename and its database file for INFOS processing on BASIC channel C.

Read a Record or a Partial Record (READ)

Format:

NFOS READ E, *[C,B$,L],* D$,P$; posspec

Where:

L is the lock/unlock specification.

Values for *L* are:

0 No locking or unlocking
1 Lock the database record
2 Lock the index entry
3 Lock both the database record and the index entry
-1 Unlock the database record
-2 Unlock the index entry
-3 Unlock both the database record and the index entry

D$ is a string which receives the database record. You can omit D$ if you do not read a database record with this request. If you omit D$, no locking or unlocking of the database will occur.

P$ is a string which receives the partial record (if any).

You may omit P$ and the preceding comma if you do not need to read the partial record. If you omit P$ and D$, then neither comma is required (the semicolon follows the right bracket). READ returns PRSRS and PRSIL.

Reinstate a Logically Deleted Record (REINS)

Format:

NFOS REINS E, *[C,B$,L];* posspec

Where:

L *is the local/global selection flag.*

Values of L are:

1 Reinstate the database record
2 Reinstate the index entry
3 Reinstate both the database record and the index entry

You can use this statement to reinstate a logically deleted database record and/or index entry.

Retrieve a Subindex Definition (RETDF)

Format:

NFOS RETDF E, *[C,B$],* KL,PL,NS;posspec

The subindex definition you indicate is returned to KL, PL, and NS (see DFSI). RETDF returns SDRMF and SDIFL.

Retrieve the Highest Key in an Index (RETHK)

Format:

NFOS RETHK E, *[C,B$],* K$;posspec

The value of the highest key in the index or subindex positioned to is returned in K$.

Retrieve a Key (RETKY)

Format:

NFOS RETKY E, *[C,B$],* K$;posspec

The value of the key for the entry positioned to is returned in K$.

Licensed Material-Property of Data General Corporation

Retrieve Record Status (RETST)

Format:

NFOS RETST E, *[C,B$]*, posspec

You can use this statement to retrieve the status of the record and index entry positioned to. RETST returns PRLEN, PRSRL, and PRSRS.

Rewrite a Record (REWRITE)

Format:

NFOS REWRITE E, *[C,B$,L]*, D$,P$;posspec

All of the arguments for rewrite are the same as those for WRITE.

Write a Record (WRITE)

Format:

NFOS WRITE E, *[C,B$,L]*, D$,P$;posspec

All arguments for WRITE are the same as those for READ except in a WRITE D$ may be replaced by an asterisk (*) to indicate an inverted write. The data record feedback is taken from B$ which you must change or else it will be pointing to the last record you accessed using B$ as the buffer string. The WRITE statement uses PRRMF and PRDFB (if it is an inverted write) and returns PRDFB and PRSIL.

Packet Definitions

Following are definitions for fields in the buffer string which you may wish to examine or modify for certain functions. To examine an item, use ASC (B$[i,j]). To test a bit flag, use AND (ASC (B$[i,j]), bit-value). To set a new value use B$[i,j] = CHR$ (new-value,j-i+1). To set a bit flag, use B$[i,j] = OR (ASC(B$[i,j]), bit-value).

B$[7,8]	FDFL1	File definition flags 1

For the OPEN function:

1024	F1RAW	Read after write verification

For all other functions:

B$[1,2]	PRSTA	Processing status flags
1024	PFWIF	Write immediately if modified
32	PFPEF	Physical end of file
16	PFVER	Verification failure
B$[11,12]	PRLEN	Record length
B$[17,20]	PRSIL	Subindex level
B$[22,22]	PRRMF	Record merit factor
B$[37,37]	PRSRL	System returned length
B$[38,38]	PRSRS	System returned flags
256	SRLLD	Local logical delete
128	SRDUP	Duplicate key
64	SRDRL	Data record locked
32	SRGLD	Global logical delete
B$[52,52]	SDRMF	Root node merit factor
B$[53,54]	SDIFL	Index flags
32768	IXPKC	Perform key compression
16384	IXNSI	No subindexes
8192	IXHPN	High priority node
4096	IXTSI	Temporary subindex
2048	IXPRM	Permanant data record

Licensed Material-Property of Data General Corporation

INFOS Example

```
0010 DIM X$(200),C$(100),K$(10),ANS$(1)
0020 NFOS OPEN (0,X$),"INDEX1"          :Open INDEX1 to database.
0030 NFOS OPEN (1,X$),"INDEX2"          :Open INDEX2 to database.
0040 LET X$=FILL$(0)                    :Initialize buffer.
0050 INPUT "DEPOSIT= ",DEP              :Enter a deposit amount.
0060 INPUT "KEY= ",K$                   :Enter a key to update.
0065 IF K$(1,3)="END" THEN GOTO 0600    :Input complete?
0070 IF LEN(K$)<10 THEN GOTO 0200       :Was key complete?
0080 NFOS READ (0,X$,1),C$;;K$          :Read data and lock.
0110 LET BAL=ASC(C$(11,14))             :Extract balance from record.
0120 LET BAL=BAL+DEP                    :Upcate balance.
0130 IF BAL>10000 THEN GOSUB 0500       :Write new record if BAL>$100.00.
0140 LET C$(11,14)=CHR$(BAL,4)          :Upcate balance in record.
0150 NFOS REWRITE (0,X$,-1),C$;8        :Return record and unlock.
0180 GOTO 0040

0200 REM --PARTIAL KEY                  :Partial generic key.
0210 NFOS RETKY ER,(0,X$),C$;;K$,2      :Read index with generic.
0220 IF ER=0 THEN GOTO 0260             :Error?
0230 IF ER<>-190 THEN GOTO 9000         :Key position error?
0240 PRINT "NO GENERIC KEY FOUND."
0250 GOTO 0040
0260 PRINT "FULL KEY= ";C$              :Display the key found.
0270 INPUT "IS THIS CORRECT? ",ANS$
0280 IF ANS$="Y" THEN GOTO 0400         :If yes, go read; if no, try again.
0300 NFOS RETKY ER,(0,X$),C$;1          :Read next key.
0310 GOTO 0220

0400 NFOS READ (0,X$,1),C$;8            :Read the data and lock the record.
0410 GOTO 0110                          :Process the record.

0500 REM -- WRITE NEW RECORD           :To write all records with balances
0510 NFOS WRITE (1,X$),C$;-9;C$(1,10)   :over $100.00 to second file.
0520 RETURN

0600 CLOSE                              :Close all files.
0610 END

9000 LET C$=ERM$(ER)
9010 PRINT C$                           :Print error message.
9020 GOTO 0600

* RUN )
DEPOSIT= 1200 )
KEY= ABC1234567 )
DEPOSIT= 100 )
KEY= ABC1234 )
FULL KEY=ABC1234560
IS THIS CORRECT? N )
FULL KEY=ABC1234563
IS THIS CORRECT? Y )
DEPOSIT= 1000 )
KEY= AAA )
NO GENERIC KEY FOUND
DEPOSIT= 0 )
KEY= END )
*
```

End of Chapter

Licensed Material-Property of Data General Corporation

Chapter 10
System Management

This chapter outlines the priviliged information for system managers and system operators, and it should only be read by people who are managing a Business BASIC system.

Business BASIC is designed to run on RDOS or DOS operating systems. In multiuser environments you should use RDOS, and if you want to use the INFOS interface you need an RDOS INFOS system. You must first generate your operating system before generating Business BASIC.

When generating an RDOS or DOS system, follow the instructions included with your operating system, and refer to the proper reference manual.

Business BASIC has an .IDEF start-up sequence to generate asynchronous (ALM) driver support and the second TTY ($TTO1/$TTI1) support. A conflict would arise if the operating system includes ALM or the second TTY support. You can generate these for your operating system if Business BASIC will not use them. Otherwise, do not generate these with the operating system; leave it to Business BASIC. If you are running Business BASIC in the foreground on a console, you must generate a second TTY in the operating system.

You should generate your RDOS system with at least three stacks. You should also generate as many channels as you need for Business BASIC users in whatever ground Business BASIC will be running. Use tuning wherever possible to determine whether or not your resources (stacks, cells, buffers, etc.) need modification.

You'll also need to access at least four subdirectories simultaneously, so you must generate the capability into the operating system. We recommend you also create a subdirectory for each user.

Business BASIC supports more than two users by employing asynchronous multiplexors.

Installation -- RDOS

For RDOS based systems, we supply Business BASIC on magnetic tape or on diskette; all files are in RDOS dump format. To install Business BASIC into an RDOS system, follow these procedures:

Loading from Magnetic Tape

Make the current directory be the primary or secondary partition where you want the Business BASIC system to reside, then use these RDOS CLI commands:

```
INIT MTx
```

where x is the drive number where you mounted the tape.

```
LOAD/V MTx;0
RELEASE MTx
```

Loading from Diskettes

For each diskette, use the following RDOS CLI commands, with the current directory set as the partition you want Business BASIC to reside:

```
INIT DPx
```

where x is the drive number where you mounted the diskette.

```
LOAD/V DPx:filename
```

where filename is as follows:

$SYSDR.DP	for the system library diskette.
$LIBDR.DP	for the utility library diskette.
$DOCDR.DP	for the utility source diskette.
NBASICDR.DP and $GAMESDR.DP	for games and other programs.
BASICDR.DP	for the build modules diskette.

Licensed Material-Property of Data General Corporation

Installation -- DOS

We supply Business BASIC on diskette in directory format for DOS systems. Generate the DOS system with files RLDR.SV, RLDR.OL, MAC.SV, and MACXR.SV on the DOS system diskette. Run DOS on diskette 0, and mount the BASICGEN diskette on DP1. Use INIT to initialize DP1. Move necessary files to the BASICGEN diskette by typing:

```
MOVE/V DP1 RLDR.SV RLDR.OL MAC.SV MACXR.SV )
```

Delete all unnecessary files from DP0. The only files you need are BOOT.SV, CLI.ER, CLI.OL, CLI.SV, DOS-system-name.SV, and DOS-system-name.OL.

Generate DOS Business BASIC by typing:

```
DIR DP1 )
BSG DP0:systemname )
```

Answer the BSG questions (described next in this chapter). Then return to DP0, print the listing file and load map, and save them for future reference before deleting them from the system.

To create the subdirectories for Business BASIC, mount the $SYS and $LIB diskette on DP1, then type:

```
INIT DP1 )
DIR DP1 )
MOVE/V DP0 -.DR )
$SYS
$LIB
RELEASE DP1 )
```

At this point, your Business BASIC system is running on the system diskette. Files contained in $SYS and $LIB are sufficient for a minimal runtime environment. Other utilities and files are supplied on additional diskettes.

Generation and BSG

To generate a new Business BASIC system, execute the RDOS/DOS BSG program by typing:

BSG *[systemname]*

Global Switches

/N Do not automatically assemble or load the system. Leave the output in the files CLI.CM, *systemname.SG,* and *systemname.CM.*

/L Print the listing file *(systemname.LS)* after the load is complete.

If you omit *systemname,* BASIC is the system name. The system generation dialog will be output to *systemname.SG,* the RLDR command line is in *systemname.CM,* and the load map listing is in *systemname.LS.* You can include a directory specifier in *systemname.*

The BSG program outputs a message and asks a series of questions to determine the parameters for the system. An outline of the dialog is shown in Figure 10-1.

We number the questions in Figure 10-1 for your convenience; no such numbering occurs during execution. You can give a default response by pressing carriage return. Except where noted, the default response for numeric questions is zero and for yes or no questions (Y or N) is no (N).

Licensed Material-Property of Data General Corporation

1. *(COMPUTER TYPE (0=MICRONOVA 1=UNMAPPED NOVA 2=MAPPED NOVA 3=ECLIPSE)?*

   Respond with the appropriate number for your computer. If you have an unmapped ECLIPSE computer, you can specify it as an unmapped NOVA computer.

   a. *SWAPPING SYSTEM (Y OR N)?*

      This question is only asked if your system is unmapped.

   b. *HARDWARE MULTIPLY/DIVIDE (Y OR N)?*

      This question is only asked if you have a NOVA or MICRONOVA computer. If you use the hardware multiply/divide unit, you should get about a 20% speed increase.

   c. *PARTITION SIZE (1K WORD BLOCKS)?*

      Respond with the maximum number of 1024 word pages that you want to allow Business BASIC to use. Default is 32 blocks.

   d. *WINDOW MAPPING (Y OR N)?*

      This and the following question are only asked if you're using a mapped system.

      MAXIMUM NUMBER OF VIRTUAL PAGES?

      This question is asked if your answer to 1d. is yes. Default value is 32767.

2. *DOUBLE (0) OR TRIPLE (1) PRECISION?*

   Respond with a 1 only if your system needs triple precision arithmetic. This allows a larger range (15 digits) than the usual 10 digits.

3. *CONSOLE TERMINAL (Y OR N)?*

   Respond with a yes if the operating system's console will be used. This is $TTI/$TTO when BASIC runs in the background, and &TTI1/$TTO1 when BASIC is running in the foreground.

4. *SECOND TTY (Y OR N)?*

   Respond with a yes only if you will run BASIC in the background, no foreground will be running, and the system has a $TTI1/$TTO1 (device codes 50 and 51) which has *not* been qenerated into the RDOS/DOS operating system.

   a. *4029 MODEM CONTROL (Y OR N)?*

      Respond with a yes if the foreground controller has the 4029 modem control option installed and you want to use it.

Figure 10-1. Sample BSG Dialog

Licensed Material-Property of Data General Corporation

5. *NUMBER OF MULTIPLEXOR LINES?*

   Respond with the number of lines on the multiplexor (0 to 32). If the number of lines is zero, skip to question 6.

   a. *MULTIPLEXOR TYPE (0=4060 1=ALM)?*

      Respond with the multiplexor type you have. A ULM-5 would be the same type as the ALM.

   b. *PRIMARY (0) OR SECONDARY (1)?*

      Respond with a 1 if the multiplexor is the secondary device code.

   c. *MODEM CONTROL (0=NONE 1=4026 2=ALM)?*

      Respond with the appropriate modem control type.

   d. *CHANGE DEFAULT LINE SPEED (Y OR N)?*

      If you're using ALM, you have the option of specifying the clock (0-3) to be used for each line. By default, clock 0 will be used for all lines. If you answer yes, then the following question will be asked for each line:

      CLOCK FOR LINE n?

      Respond with the clock (0-3) to be used for that line.

6. *PROCESSOR (0=SINGLE 1=MASTER 2=SLAVE 3=SECONDARY SINGLE)?*

   Respond with 1 for the master processor in a dual processor system; 2 for the slave processor in a dual processor system. Respond with a 3 if a second copy of Business BASIC will be running, but not in a shared processor environment. If you're going to run a system in the background and a system in the foreground, generate the background version as a single and the foreground as a secondary single. Record locking will not operate between the two copies.

   a. *SYSTEM DIRECTORY NAME?*

      This question is only asked if the answer to 6 is either 2 or 3. This allows setting of an alternate directory to $SYS so that conflicts do not occur over the BASIC.SW and BASIC.PS files (push and swap). Your directory name must be exactly four characters in length.

7. *INFOS (Y OR N)?*

   This question is only asked if you have an ECLIPSE computer. Respond with yes only if you intend to use the INFOS interface and you have an RDOS INFOS system.

8. *RUN ONLY (Y OR N)?*

   Respond with a yes only if you want a run only system. This will inhibit the loading of the compiler section of Business BASIC. In keyboard mode, you can only use the RUN command (this is emphasized by the prompt being changed to *RUN*. The ENTER and ERASE statements will not work even if you include them in programs. This will save about 2500 bytes in memory.

9. *USER WRITTEN SUBROUTINES (Y OR N)?*

   Respond wiht yes if the module USERSUBS is loaded with Business BASIC. You can then access assembly language subroutines with the UCALL statement.

Figure 10-1. Sample BSG Dialog (continued)

Licensed Material-Property of Data General Corporation

10. *MAXIMUM NUMBER OF JOBS?*

Respond with the maximum number of jobs which may be running at one time. The BASIC execution command actually controls the number, up to the number you answer with here. The default is one job.

11. *MAXIMUM PROGRAM SIZE (512-BYTE PAGES)?*

This is the maximum number of 512-byte pages that a job may have. The HELLO program may allocate up to this number of pages to a job. This parameter mainly controls the size of the swapping files (BASIC.SW and BASIC.PS). An algorithm for calculating each file is:

BASIC.SW = (pages + 1) * (number of jobs at execution time)

BASIC.PS = 3 * (BASIC.SW) (assumes 3 levels of swapping)

12. *NUMBER OF USER CHANNELS?*

This is the total number of channels which may be opened by all BASIC programs. The default is 8 times the number of jobs. Do not generate more channels than the operating system has.

13. *NUMBER OF LOCK AREAS?*

This is the total number of LOCK's that can be active simultaneously. The default is 2 times the number of jobs.

14. *INPUT BUFFER SIZE (BYTES)?*

Respond with the number of bytes that a terminal may input ahead of those which have been echoed. If too few bytes are allocated for the input buffer, you might lose characters typed by a fast typist. A good number is 50. The default is 10 bytes.

15. *OUTPUT BUFFER SIZE (BYTES)?*

Respond with the number of bytes that will be buffered during output. The larger this buffer, the more a program may overlap with output to its own terminal. Default is 20 bytes.

16. *CHANGE DEFAULT TERMINALS (Y OR N)?*

By default, subroutines to handle cursor positioning and terminal characteristics are included for terminal types 0 through 9. If you want, you can load only the subroutines for the terminal types you are using. If you answer yes, the following question will be asked for each terminal type:

TYPE n (Y OR N)?

Respond with yes if you want the driver for this terminal type (loaded from TERM.LB).

Figure 10-1. Sample BSG Dialog (continued)

Licensed Material-Property of Data General Corporation

17. *CHANGE DEFAULT OVERLAYS (Y OR N)?*

    Respond with yes only if you want to load some overlays into core for improved throughput. If you answer no, skip to question 18; otherwise, answer yes to each question if you want the module to be loaded in core rather than remain overlayable:

    a. *SYNTAX TABLES (Y OR N)?*

       This question is not asked if you're using INFOS. The syntax tables are used to compile BASIC statements.

    b. *KADD, KFIND, KNEXT, KDEL (Y OR N)?*

       These are for Business BASIC ISAM (index) routines.

    c. *PRINT USING (Y OR N)?*

       This module processes PRINT USING statements.

18. *VIRTUAL OVERLAYS (Y OR N)?*

    This question is asked only if you're using a mapped system. If you answer yes, then all overlays are loaded into extended memory.

19. The following questions control the reserved file table. This table is used to prevent more than one user from opening the same device. Answer yes if that device is to be included in this reserved file table:

    a. *LINE PRINTER (Y OR N)?*

    b. *MAGNETIC TAPE DRIVE (Y OR N)?*

    c. *ADDITIONAL RESERVED FILE NAMES (Y OR N)?*

       If you answer yes to additional filenames, then the following question will be repeated until you give it a null response:

       RESERVED FILE n?

       Respond with a reserved filename which must be exactly four characters long and either start with a dollar sign ($) or end with a colon (:).

Figure 10-1. Sample BSG Dialog (continued)

After responding to the last question for BSG and assuming you didn't use the /N global switch, the system will assemble *systemname.SG* and SY.SR using the macroassembler (MAC) and output the listing to *systemname.LS*. It will will then load the system into *systemname.SV* and *systemname.OL* using RLDR.SV and RLDR.OL with the load map in *systemname.LS*. If you included the global /L switch to BSG, it will then print *systemname.LS* on the line printer.

Licensed Material-Property of Data General Corporation

Sample BSG Dialogs

These examples are only sample BSGs and are not necessarily good examples for your unique environment. The listing after the last BSG question is the load map. Each example is the output from *systemname.LS*. Example 10-1 is a generation on an ECLIPSE computer, and Example 10-2 is a generation on a MICRONOVA computer.

Example 10-1

```
;       SYSGEN FOR REV 03.10 FOR EBASIC
; COMPUTER TYPE (0=MICRO NOVA 1=UNMAPPED NOVA 2=MAPPED NOVA 3=ECLIPSE)? 3
; PARTITION SIZE (1K WORD BLOCKS)? 32
; WINDOW MAPPING (Y OR N)? Y
;    MAXIMUM NUMBER OF VIRTUAL PAGES? 32767
; DOUBLE (0) OR TRIPLE (1) PRECISION ? 1
; CONSOLE TERMINAL (Y OR N)? Y
; SECOND TTY (Y OR N)? N
; NUMBER OF MULTIPLEXOR LINES? 7
;    MULTIPLEXOR TYPE (0 = 4060  1 = ALM)? 0
;    PRIMARY (0) OR SECONDARY (1)? 0
;    MODEM CONTROL (0 = NONE  1 = 4026  2 = ALM)? 0
; PROCESSOR (0 = SINGLE  1 = MASTER  2 = SLAVE  3 = SECONDARY SINGLE)? 0
; INFOS (Y OR N)? Y
; RUN ONLY (Y OR N)? N
; USER WRITTEN SUBROUTINES (Y OR N)? N
; MAXIMUM NUMBER OF JOBS? 7
; MAXIMUM PROGRAM SIZE (512 BYTES PAGES)? 0
; NUMBER OF USER CHANNELS? 16
; NUMBER OF LOCK AREAS? 14
; INPUT BUFFER SIZE (BYTES)? 10
; OUTPUT BUFFER SIZE (BYTES)? 20
; CHANGE DEFAULT TERMINALS (Y OR N)? N
; CHANGE DEFAULT OVERLAYS (Y OR N)? N
; VIRTUAL OVERLAYS (Y OR N)? Y
; LINE PRINTER (Y OR N)? Y
; MAGNETIC TAPE DRIVE 0 (Y OR N)? N
; ADDITIONAL RESERVED FILE NAMES (Y OR N)? N
        .TITLE  EBASIC  ;EBASIC
        NJBS    =       7.
        NMUX    =       7.
        DUALP   =       0.
        .ENT    PROCD   ; PROCESSOR NUMBER
        PROCD   =       0.
        NLOCKS  =       14.
        .ENT    IPRCSN  ; PRECISION FLAG
        IPRCSN  =       1.
        MCHAN   =       21.
        ICHAN   =       4.
        RCHAN   =       5.
        CONSOLE =       1.
        FTTY    =       0.
        PAGES   =       0.
        ?ADSZ   =       32.
        WMAP    =       1.
        ?VMSZ   =       32767.
        .ENT    MPDC    ; MULTILPEXCR DEVICE CODE
        MPDC    =       24.
        .ENT    SYS?1 SYS?2     ; SYSTEM DIRECTORY NAME
        SYS?1   =       '$S'
        SYS?2   =       'Y1'
        CLOCK   =       1.
        FMUX    =       0.
        FMOD    =       0.
        F29     =       0.
        FNFOS   =       1.
```


Licensed Material-Property of Data General Corporation

Example 10-2

```
;       SYSGEN FOR REV 03.10 FOR MICBASIC
; COMPUTER TYPE (0=MICRO NOVA 1=UNMAPPED NOVA 2=MAPPED NOVA 3=ECLIPSE)? 0
; SWAPPING SYSTEM (Y OR N)? N
; HARDWARE MULTIPLY/DIVIDE (Y OR N)? N
; PARTITION SIZE (1K WORD BLOCKS)? 32
; DOUBLE (0) OR TRIPLE (1) PRECISION ? 1
; CONSOLE TERMINAL (Y OR N)? Y
; SECOND TTY (Y OR N)? N
; NUMBER OF MULTIPLEXOR LINES? 7
;    MULTIPLEXOR TYPE (0 = 4060  1 = ALM)? 0
;    PRIMARY (0) OR SECONDARY (1)? 0
;    MODEM CONTROL (0 = NONE  1 = 4026  2 = ALM)? 0
; PROCESSOR (0 = SINGLE  1 = MASTER  2 = SLAVE  3 = SECONDARY SINGLE)? 0
; RUN ONLY (Y OR N)? N
; USER WRITTEN SUBROUTINES (Y OR N)? N
; MAXIMUM NUMBER OF JOBS? 7
; MAXIMUM PROGRAM SIZE (512 BYTES PAGES)? 0
; NUMBER OF USER CHANNELS? 16
; NUMBER OF LOCK AREAS? 14
; INPUT BUFFER SIZE (BYTES)? 10
; OUTPUT BUFFER SIZE (BYTES)? 20
; CHANGE DEFAULT TERMINALS (Y OR N)? N
; CHANGE DEFAULT OVERLAYS (Y OR N)? Y
;    SYNTAX TABLES (Y OR N)? Y
;    KADD, KFIND, KNEXT, KDEL (Y OR N)? Y
;    PRINT USING (Y OR N)? Y
; LINE PRINTER (Y OR N)? Y
; MAGNETIC TAPE DRIVE 0 (Y OR N)? Y
; ADDITIONAL RESERVED FILE NAMES (Y OR N)? N
        .TITLE  MICBASIC        ;MICBASIC
        NJBS    =       7.
        NMUX    =       7.
        DUALP   =       0.
        .ENT    PROCD   ; PROCESSOR NUMBER
        PROCD   =       0.
        NLOCKS  =       14.
        .ENT    IPRCSN  ; PRECISION FLAG
        IPRCSN  =       1.
        MCHAN   =       22.
        ICHAN   =       4.
        RCHAN   =       6.
        CONSOLE =       1.
        FTTY    =       0.
        PAGES   =       0.
        ?ADSZ   =       32.
        WMAP    =       0.
        ?VMSZ   =       32767.
        .ENT    MPDC    ; MULTILPEXOR DEVICE CODE
        MPDC    =       24.
        .ENT    SYS?1 SYS?2     ; SYSTEM DIRECTORY NAME
        SYS?1   =       '$S'
        SYS?2   =       'Y1'
        CLOCK   =       54.
        FMUX    =       0.
        FMOD    =       0.
        F29     =       0.
        FNFOS   =       0.
        FUSER   =       0.
```


Licensed Material-Property of Data General Corporation

Example 10-2 (continued)

```
.UISZ   =       5.
.UOSZ   =       10.
RFN     =       2.
N1001   =       '$L'
N2001   =       'PT'
N1002   =       'MT'
N2002   =       '0:'
FTRM0   =       1.
FTRM1   =       1.
FTRM2   =       1.
FTRM3   =       1.
FTRM4   =       1.
FTRM5   =       1.
FTRM6   =       1.
FTRM7   =       1.
FTRM8   =       1.
FTRM9   =       1.
C?000   =       0.
C?001   =       0.
C?002   =       0.
C?003   =       0.
C?004   =       0.
C?005   =       0.
C?006   =       0.


FUSER   =       0.
.UISZ   =       5.
.UOSZ   =       10.
RFN     =       1.
N1001   =       '$L'
N2001   =       'PT'
FTRM0   =       1.
FTRM1   =       1.
FTRM2   =       1.
FTRM3   =       1.
FTRM4   =       1.
FTRM5   =       1.
FTRM6   =       1.
FTRM7   =       1.
FTRM8   =       1.
FTRM9   =       1.
C?000   =       0.
C?001   =       0.
C?002   =       0.
C?003   =       0.
C?004   =       0.
C?005   =       0.
C?006   =       0.
```


Licensed Material-Property of Data General Corporation

Executing Business BASIC

To execute Business BASIC once you've generated it, type the system name of your Business BASIC system:

```
systemname n/J
```

The /J local switch appended to n indicates that you are allowing n number of jobs to run simultaneously in your Business BASIC system. If you don't specify n with the /J local switch, you will only be able to run one job.

If you include the /D global switch appended to systemname, you will enable RDOS/DOS interrupts to stop the execution of your Business BASIC system. This is only recommended for single user systems, because if you enable the RDOS/DOS interrupts, any user that happens to hit the RDOS/DOS interrupt keys (CTRL-A, CTRL-C, CTRL-F) will bring down the Business BASIC system abruptly. Since you can always use KILL or QUICKILL to bring down the Business BASIC system, you should not use the /D global switch unless you are bringing up a temporary system.

Here's an example (the *R* is the DOS/RDOS CLI prompt):

```
R
BBASIC )
```

This executes the file BBASIC.SV to bring up the Business BASIC system and allows BBASIC only one job.

```
R
BBASIC 5/J )
```

This brings up BBASIC and allows five jobs to run simultaneously on the system. If you have five terminals on line, each one will sound a bell to indicate that they are ready for logging on to Business BASIC.

When you execute Business BASIC, the message:

DGC BUSINESS BASIC REV. xx.xx

appears. Then you can hit the ESCape key to start logging on.

Generating the Push Space

There must be sufficient space in the file BASIC.PS in $SYS (or $SY1 for a system with triple precision) for the Business BASIC SWAP statement to work. By default, this file is made to be three times the size of BASIC.SW (swapping file), which is enough space for three SWAP statements (two nested within the first). If you want a greater nesting limit for SWAPs, you must CCONT the file $SYS:BASIC.PS to a size size specified by this formula:

```
(max pages per job + 1) * (max number of SWAPs)
  * (max number of jobs)
```

System Security

Business BASIC includes extensive security and file protection features to meet the following security needs: protecting the system from outsiders, protecting users from each other, and protecting and preserving the integrity of the programs and data.

HELLO and Accounts

To protect the system from unauthorized users, Business BASIC maintains a password-account system using the HELLO program and the ACCOUNTING file. HELLO is the BASIC program that handles logging on. A standard HELLO program is provided in the directory $SYS, which only the system manager (AA account) can access. You can change your HELLO program and tailor it to your installation, or you can use the standard HELLO provided, but you *must* modify your ACCOUNTING file to permit new accounts. You modify ACCOUNTING using File Maintenance (see Chapter 5), and specifying ACNTS as your data file (ACNTS.TB is the table file).

When a user logs on, the standard HELLO program determines the terminal type, or asks for the terminal type if it does not recognize the terminal. If you use non-DG terminals, you should ask your DG representative for the proper type. DG 6053 terminals are recognized by HELLO (as terminal type 6), so if you log on with a 6053, the question is not asked. If you use a DG 6012 or similar terminal, respond with terminal type 3. Hard copy terminals are type 2.

Licensed Material-Property of Data General Corporation

HELLO then asks for an account ID and a User ID. These two IDs are combined to form a single user's account ID. The account ID is a two-letter code designating the type of account: AA is the system manager, AB is a system programmer, OP is the system operator, and you can make up more codes for other types of accounts. The User ID is a three-letter code unique for each user (usually the user's initials). HELLO then asks for a password, which is not echoed when the user types it. HELLO then checks the account ID against the password based on the information in the ACCOUNTING file. If the account ID does not agree with the password or if HELLO can't find the account ID in the ACCOUNTING file, the log on is terminated, and the user must try again.

Business BASIC provides a special password coded into the HELLO program for the system manager, and passwords for the system programmer and the system operator in the ACCOUNTING file. The system manager's (AA account) password is DE2LA6; the system programmer's (AB account) password is DE2ACE; and the system operator's (OP account) password is DEG2YQ. You should change these passwords once you've generated your Business BASIC system -- just LIST the HELLO program (in the $SYS directory) and change the appropriate string assignment for the password, and use File Maintenance to change the ACCOUNTING file information.

Finally, HELLO may or may not ask for a directory specifier. If you set a specific directory for a user in the ACCOUNTING file, the question will not be asked of that user. If you did not specify a directory for that user, the question is asked; if the user specifies a directory he/she is restricted from, log on is terminated, and the user must try again. You can make an account ID always sign on to a specific directory by specifying the directory name for that account in the ACCOUNTING file. You can also restrict a directory to specific users (even if other users are allowed to DIR to any directory) by using a U$ERS file in the directory that controls the access to that directory.

ACCOUNTING

Exept for the AA account group, passwords and account IDs are stored in the ACCOUNTING file in directory $SYS (only AA accounts have access to $SYS). ACCOUNTING can be changed and/or expanded using File Maintenance (see Chapter 5). You can maintain ACCOUNTING using FM, and a table file (ACNTS.TB) is already supplied. As an exercise, if you are an AA account, get into $SYS and RUN "FM", specifying ACNTS as your filename. Using FIND and FIND NEXT, step through the ACCOUNTING file where you can see the accounts already supplied.

In the ACCOUNTING file, you can set up a list of privileges for each user. These include the user's job priorities, the maximum amount of memory a user's program can use, an optional directory and/or program in which the user starts, whether or not the user is allowed to change directories, and whether the user can use all, some or none of the BASIC CLI commands.

For any user, you can set a valid account ID/password in the ACCOUNTING file. You can also specify a directory you want the user to work from, or leave the field blank in which case HELLO will ask the user for a directory. You can specify a program you want to run initially for the user (e.g., a menu program, discussed next), and when the user logs on, this program will immediately run (useful for run-only users -- you supply a menu program); or leave it blank and the user will log on the BASIC's keyboard mode. You can allow the user to change directories with a 1 for the ADIR flag, or leave it blank and the user will not be able to change directories.

You can allow use of the BASIC CLI CHATR command to change file attributes, and allow use of the CLI START command to start de-attached jobs, allow use of the BASIC CLI itself, by specifying a 1 for each field (or leave any blank and they will not be allowed). You can inhibit messages to the user, inhibit execution of the NEWS program (see NEWS in this chapter), and allow an automatic log off (timeout) after 10 minutes of inactivity (useful if you have remote users). You can also set the I/O priority, the compute priority, the disk priority, the maximum number of pages (core), and the output queue name (usually not set -- output goes to line printer). If your user plans to use FM and/or SM, you should give the user at least 24 pages of core.

If you set the forced SWAP flag, it will only work for the initial program you specify for the user, and the program will be SWAPped to instead of CHAINed, so that the user can only run that program. You can make the program a menu program (see "The Menu Concept") that SWAPs to other programs and contains on ON IKEY THEN BYE, so if the user hits an interrupt key, he/she will be logged off the system (this prevents the use of LIST, LOAD of new programs, etc.) You can also constrain all users by answering YES to the BSG question RUN ONLY? and using the BASIC CLI START command (see Chapter 8) to produce a system that does not allow program development or tampering with programs.

Example 10-3 shows an access to ACCOUNTING using roll-mode FM. We are adding a record for the GM account, which has no password, that logs on in directory $GAMES with the program GM. There is a forced SWAP to GM so the user can execute any of the programs GM SWAPs to, but if the user hits certain keys or the interrupt key, the program will return the user to the GM program which in turn logs the user off the system. Use of the BASIC CLI is not allowed.

Licensed Material-Property of Data General Corporation

Example 10-3

```
*RUN "FM" )
FILENAME: ACNTS )
ACCOUNTS
FUNCTION? ADD )
REC # 4  ACNT: GM )   PWORD: )   DIRTRY:  $GAMES )   PROG. GM )
IOPRI: 0 )   COMPRI: 0 )   DSKPRI: 0 )   ADIR: )   ACHATR: )

TIMEOUT: 1 )  NOMSG: )   FSWAP: 1 )   ACLI: )   START: )
NONEWS: )   NPAGES: 24 )  OUTQUE: )
FUNCTION? STOP )
*
```

No one using the GM account needs a password to gain access. Users are automatically put in $GAMES and the GM program (a menu program) asks what game they want to play. The person using the GM account must respond in ten minutes or they are logged off. They can receive messages and NEWS, and their output file is the line printer.

The ACCOUNTING file can be maintained using FM. However if you need to access the file using your own program, the ACCOUNTING file's format description is shown in Table 10-1.

Licensed Material-Property of Data General Corporation

Table 10-1. ACCOUNTING File Format

Record size: 128 bytes
File Size: 1000 records
Index file: ACNTNDX
Key size: 6 bytes (5 used)

Field Description	Location	Size	Type
Status (value=1)	0	2	Integer
Account ID (key in ACNTNDX)			
Account group	2	2	String
User ID	4	3	String
unused (terminal type)	7	1	
Password	8	6	String
Directory	14	20	String
Program	34	14	String
Flags (status word 2)*	48	2	Integer
I/O Priority	50	1	Binary
Compute Priority	51	1	Binary
Disk Priority	52	1	Binary
Maximum core (pages)	53	1	Binary
Output queue name (blank for $LPT)	54	6	String
Unused	60	68	

*Bits for flags are numbered 15-0:

Bit	Use
15	Program (if set above) will be SWAPped to instead of CHAINed (forced SWAP).
12	Allow use of the BASIC CLI START command (if bit 11 is set).
11	Allow use of the BASIC CLI (will not run unless set).
10	Inhibit execution of the NEWS program.
9	Allow use of CHATR (if bit 11 is set).
8	Allow access to all directories except $SYS.
6	Inhibit receiving messages (no messages).
5	Auto log off after ten minutes of inactivity (timeout).

All other bits are masked out and may not be set.

The Menu Concept

If you set up a program to be executed when the user signs on, this program coould be a master program that contains all the facilities and SWAPs to any other programs that the user will need. A forced SWAP to this master *menu* program allows any values to be retained when the menu is executed, and the menu can SWAP to other programs; if the user interrupted or erred in the program's execution, control returns to the menu program not to keyboard mode. The menu program should, in turn, contain an ON IKEY statement to trap any interrupts so that the user can't force control to return to keyboard mode. You can make your ON IKEY force the user off the system. Any program can be made *run only* by setting the run only flag with STMB 16 (see "Priveleged System Calls").

If you generate a run only Business BASIC system, you'll need a menu program of sorts, because run only systems do not allow the use of ENTER, LIST, or LOAD. When writing programs for run only systems, you can increase execution speed by putting all DIM statements in the beginning of the program, following them with a STOP statement, executing the program and saving it on disk with its DIMensioned values, and executing the program later by using CHAIN THEN CON -- the execution skips the DIM statements and starts executing after the STOP statement.

You can also use a menu program to monitor all newly entered data (including programs) by receiving the data in keyboard mode and starting de-attached jobs that check for data integrity without disturbing the user's keyboard mode. In this way, de-attached jobs could handle output queues, transaction logging, specialized file maintenance, etc. without the user knowing about it and explicitly coding it.

U$ERS

You can protect data files and programs in directories by restricting access to directories with a U$ERS file. The U$ERS file is a text file that contains lines of account IDs and is created and maintained by the EDIT utility. Only the account IDs specified in U$ERS are allowed in the directory. The U$ERS file is not checked by BASIC; it is checked by the CLI program whenever you use the DIR command, and checked by the HELLO program to determine whether or not you are allowed to be in the directory into which you logged on. U$ERS entries take priority over ACCOUNTING file directory specifiers, so you can't specify a directory for a user in ACCOUNTING unless the directory's U$ERS file also contains the user's account ID.

Licensed Material-Property of Data General Corporation

NOTE: if a U$ERS file does not exist in the directory, no restriction of access takes place. Also, AA accounts (system manager) can have access to any directory whether or not there is a U$ERS file in the directory.

You can use dash (-) and asterisk (*) conventions in the U$ERS entries. Here is an example of a U$ERS file:

```
!TYPE U$ERS )

AB-
PRDGC-
**XYZ-
!
```

In this example, any user in the AB account group, the user PRDGC, and the user XYZ in any account group can access the current directory. You must include a trailing dash (-) to match the terminal type which is automatically appended to the account ID (you can also restrict terminal types from your directory).

NOTE: Unless you create a permanent U$ERS file (using CHATR), anyone who has access to a directory and the EDIT utility can create a U$ERS file.

Automatic Job Execution

When you execute Business BASIC from the RDOS/DOS CLI, you can use /S global switch to automatically execute the program STARTUP in the $SYS directory. You can modify STARTUP to do any type of system housekeeping you want to implement.

The version of STARTUP we supply scans the COM.CM file built by the RDOS/DOS CLI when you execute Business BASIC (see the *RDOS/DOS CLI User's Manual,* Appendix D, for a discussion of the COM.CM file). STARTUP looks for names with the /S local switch. Any such names will have the .JB extension appended to them and they will be used as indirect commands to the BASIC CLI. The -.JB files must exist in the $SYS directory.

For example, if the files -- SHOWLOG.JB, BACKUP.JB, and CLEARLOG.JB -- exist in $SYS.DR with the following contents:

```
SHOWLOG.JB:  SLINE O/L 80/W
BACKUP.JB:   DIR DATA; MOVE/V/R DATABU;
             DIR PGMS; MOVE/V/R PGMSBU; DIR
             $SYS
CLEARLOG.JB  PRINT/H SYSLOG;
             DELETE SYSLOG;
             CRAND SYSLOG
```

then the following command:

```
BBASIC/S 8/J SHOWJOB/S BACKUP/S CLEARLOG/S )
```

executes BBASIC with 8 jobs. STARTUP, invoked by the global /S, first passes SHOWLOG.JB to the CLI and attaches the job to a terminal thus allowing the remaining STARTUP jobs to be seen. Next, STARTUP passes BACKUP.JB to the CLI to backup the DATA and PGMS directories. Note that BACKUP.JB returns to $SYS so that the remaining -.JB files can be found and executed. Finally, STARTUP executes CLEARLOG.JB to print SYSLOG, deletes it and then recreates SYSLOG for future use. After this, STARTUP logs off the system since there are no more jobs to process.

NEWS

Business BASIC has an automatic log on message program called NEWS, in $SYS, which is always executed by the HELLO program during a log on, unless the user's account record has a flag set to inhibit NEWS. NEWS maintains two files: NEWS.LG and NEWS.DS. You must create NEWS.DS and use the EDIT utility to put your daily news into it. NEWS.LG is an index file with user accounts as its key entry and the date news was last received by that account as the key's pointer. NEWS scans the NEWS.DS file for a line beginning with \DATE mm/dd/yy where mm (month) starts at the seventh character position, and the date mm/dd/yy is compared to the date found in NEWS.LG for that user.

If the news in NEWS.DS is newer than the date in NEWS.LG, the user can read the news or skip it. You can create log on messages like DO YOU WANT THE NEWS AS OF 5/26/78? where the user could answer yes or no. If the news in NEWS.DS is older than the date in NEWS.LG for that user, then NEWS assumes the user already saw the news and log on finishes. NEWS only checks dates when HELLO executes it. You can RUN "NEWS" at any time to read the news. HELLO does not attempt to run NEWS for STARTed jobs.

NEWS.DS resides in the $LIB directory an you maintain it with the EDIT utility. NEWS expects the latest (newest) news to be at the beginning of the NEWS.DS file, with older news following it. Each news item can run for a number of lines, but you must start each news item with a line beginning with a backslash (\) as follows:

```
\DATE mm/dd/yy
```

where mm/dd/yy are numerics for the month, day, and year.

Licensed Material-Property of Data General Corporation

OPCLI

OPCLI is a utility for the system operator patterned after the BASIC CLI. The OPCLI tunes the system, sends messages, interrupts other jobs, and forces users off the system. Certain commands may only be performed by AA (system manager) and OP (system operator) accounts. You can use a modified OPCLI as a menu program for the operators' log on procedure. The OPCLI commands are ALL, FALL, BYE, CSLICE, IKEY, ISLICE, MSG, FMSG, PRI, SHOW, and SYSDEC. The OPCLI prompt is a right angle (>).

ALL and FALL

Format:

ALL "message"
FALL "message"

where message is the message you want to send.

These commands send a message to all terminals that are logged on. FALL sends a message even if the user has set a no message flag. A global /A switch sends a message to all terminals even if they are not logged on the system. For example,

```
>FALL/A "IF YOUR TERMINAL IS ON, YOU'LL SEE
  THIS.")
```

will send that message to every terminal hooked up to the system, whether or not they are logged on.

BYE

Format:

BYE *[jobnum [jobnum ...]]*

where *jobnum* is a job number for any account other than an AA (system manager) account. This command forces a BYE (log off) for any *jobnum* except a system manager's job. If you specify no *jobnum*, it logs you off the system.

CSLICE

Format:

CSLICE *[time]*

where *time* is the compute bound time slice you want to set in milliseconds. If you omit *time*, CSLICE displays the current value of your compute bound time slice. Here's an example:

```
>CSLICE )

COMPUTE BOUND TIME SLICE = 600
  MILLISECONDS
>
```

IKEY

Format:

IKEY jobnum *[jobnum ...]*

where jobnum

is any job number. This command interrupts a job whether or not the job has set an IKEY disable or an ON IKEY trap. Use this command to remove jobs from infinite loops.

ISLICE

Format:

ISLICE *[time]*

where *time* is the I/O bound time slice you want to set in milliseconds. This command will set the I/O bound time slice to *time* milliseconds. If you omit *time*, it displays the current value of your I/O bound time slice. Here's an example:

```
>ISLICE )
INTERACTIVE TIMESLICE = 300 milliseconds
>
```

MSG and FMSG

Format:

MSG terminal "message"
FMSG terminal "message"

These commands send message to the terminal number, terminal. FMSG forces the message to terminal even if the user has set no message flag. A /W global switch suspends you for 30 seconds to wait for a response from terminal. Here's an example:

```
>FMSG 1 "WHO ARE YOU?" )

FROM ABBBB6: THIS IS TLB
>
```

SHOW

Format:

SHOW jobnum *[jobnum ...]*

This command displays some information about the current state of jobnum. The /R global switch shows jobnum repeatedly until you hit the interrupt key, used to note any immediate changes to jobnum. See Example 10-4.

Licensed Material-Property of Data General Corporation

Example 10-4

```
>SHOW 1 )

JOB=1 IN DISK XQT PRI=13 DIR=USER PROG=FM ACNT=ABBBB6
TERM=1 SWAP=1 SIZE=10022 CPU=19.4 I/O=1271 LOGON=13:59 020102 001425
ADIR ACHATR ACLI ASTART
>
```

This shows that Job 1 is executing a disk program, with priority 13, in directory USER, running the FM program, with 1 nested SWAP. The program size is 10022 bytes. The CPU usage is 19.4 seconds. The I/O usage (number of system calls) is 1271. Account ABBBB6 logged on at 13:59, and is allowed to change directories, use CHATR, the CLI, and START. The user table address is 020102 001425.

SYSDEC

Format:

SYSDEC *[time]*

This command sets the system call time decrement to *time* milliseconds. If you omit *time*, it displays the current value of the system call time decrement. Here's an example:

```
>SYSDEC )

SYSTEM CALL DECREMENT=20 MILLISECONDS
```

PRI

Format:

PRI jobnum *[priority/switch]*

Switches:

/C Compute priority
/I Interactive priority
/S System call priority

This command will display/set the priority of jobnum. If you specify jobnum without a *priority/switch*, PRI will display the priority of the job. If you specify jobnum with a *priority/switch*, PRI will set the priority number in *priority* for the *switch* specified for the job jobnum. Here's an example:

```
>PRI 1 13/C )
```

This sets job 1's compute priority to 13.

OPCLI users can also execute the BASIC CLI commands STAT and TPRINT (see Chapter 8 for these commands).

NOTE: you can also execute KILL from the OPCLI, but if you're not signed on as an AA (system manager) account, KILL first asks for the AA password, and does not execute until it gets the right password.

Privileged System Calls

Business BASIC gives the system manager the ability to access any portion of memory and alter memory contents, interrupt and log off jobs, tune system resources, execute RDOS/DOS system calls and special programs, and send messages to all terminals. These tasks are accomplished using the privileged system calls STMB, STMC, and STMD.

STMB

STMB allows you nearly indiscriminate modification of memory. You must be extremely careful when you alter memory contents making sure that you are actually modifying the locations you want. The general format of the STMB statements is:

STMB call, arguments

where call is the number of the STMB call and arguments vary with each call. We discuss each STMB call separately below.

STMB 0, item, address

where address is a variable receiving the memory address of item, and item is one of the following:

0 address of table containing all of the following:
1 line table address.
2 job table address.
3 highest line
4 interactive time slice.
5 compute bound time slice.
6 channel table.
7 address of reserved filename table.
8 maximum memory allowed.
9 global switches.
10 number of jobs.
11 system call decrement.
12 lock buffers.
13 overlay statistics.
14 length in blocks of user in swap files (BASIC.SW and BASIC.PS).
15 double word counter used by SYS(29).

Licensed Material-Property of Data General Corporation

STMB 1, wordcount, address, destination

Copies the contents of memory starting at address (word address) for the number of words specified in wordcount to the variable or array destination.

STMB 2, wordcount, address, source

Copies the specified number of words in wordcount from the variable or array source into memory at the address (word address) you specified.

STMB 3, byteaddress, destination$

Copies bytes from memory beginning at byteaddress (word address * 2) into the string variable or substring destination$ until destination$ is full.

STMB 4, byteaddress source$

Copies the entire contentss of string variable or substring source$ into memory starting at the address specified in byteaddress (memory address * 2).

STMB 5 not used.

STMB 6, jobnum, error

Attaches the specified de-attached job jobnum to the current terminal, returning a possible error in error. The possible errors you may receive are:

70 invalid job number.
51 job not logged on.
71 job already attached (must be de-attached).

STMB 7, jobnum, error

Forces the specified job jobnum to BYE off the system, returning a possible error in error. The possible errors you may recieve are:

0 no error.
70 invalid job number.
51 job not logged on.

STMB 8, jobnum, error

Resets the ON IKEY condition for the specified job jobnum, clears the ignore IKEY flag, and then simulates an IKEY, stopping the program. Possible errors returned in error are:

0 no error.
70 invalid job number.
51 job not logged on.

STMB 9 not used.

STMB 10, func, address, mask

Sets (func =1) or clear (func=0) bits in the word at the address specified according to bits set in the variable mask. This is performed with the interrupts off.

STMB 11, string, var_1, var_2, var_3

Scans the contents of string tallying all bits of value 1. This count is returned in var_1. The longest substring of bits of value 0 is returned in var_2. The sum of the string's input value and var_2 is returned in var_3.

STMB 12, jobnum

Scans the job table for the first available job and executes HELLO for that job. The number of the job is returned in jobnum, or if no available jobs exist, a -1 is returned in jobnum.

STMB 13, jobnum, error, flag, string

Places the contents of string into the input buffer for the specified job jobnum. The SYS(10) value for jobnum is set to the value of flag (0 or 1), and the job is unpended. If the job is not waiting for input or doesn't exist, an error will be returned in error.

STMB 14, jobnum, error, flag

The SYS(10) value for jobnum is set to the value of flag (0 or 1), and the job acts as if an interrupt occurred. If the job doesn't exist, an error is returned in error

STMB 15, linenum, char

The ALM line linenum is set to the line characteristics in char. The first ALM linenum is 2. If char does not have 1BO (32768) set, then the current line characteristics are returned to char. For invalid lines or non-ALM multiplexors, char is set to -1.

STMB 16, flag

Set the run only flag in the current program to the value of flag (0=not set, 1=run only). This flag prevents a file from being LOADed or modified by non-AA (system manager) accounts. Also, when the program is run, IKEY's are disabled and an STMA 7,5 is executed in the program after the ON IKEY handler has been set up.

STMB 17

Shuts down the ALM multiplexor and executes the .RTN RDOS/DOS system call (used by KILL).

Licensed Material-Property of Data General Corporation

STMB 18, address, value

Returns to value the word stored at address in the operating system's address space.

STMB 19 not used.

STMB 20

Completes the housekeeping in the Business BASIC system needed to BYE the current job off the system.

STMC

Business BASIC allows the system manager (AA account) to code programs using RDOS/DOS system calls. Before you use STMC's you have to be familiar with RDOS/DOS system calls in the *RDOS Reference Manual* and the *DOS Reference Manual*. You must read these manuals for descriptions of the system calls, the arguments you need to pass to the system calls, and the possible errors returned by the system calls.

The STMC statement translates directly into an RDOS/DOS system call, only checking for compatibility of variable types. No checking is done to see if the variables you use for returning values are large enough. Therefore, if a 36-byte string is required and only a 20-byte string is supplied, the 16 bytes following the string will be overwritten.

When you use a string literal as an argument, the literal must include a terminating null. Without the null, RDOS/DOS reads beyond the end of the literal producing erroneous results. In order to pass the literal ABC, you must write the literal as ABC<0>. When you use a string variable to return an argument (in .STAT, .RSTAT, .CHSTS), you should fill the variable with nulls before calling the STMC. These STMC's do not alter the current length of the string variable; thus, if a filename were returned to a string with a current length less than the length of the filename, the characters beyond the current length would be inaccessible.

Arguments which refer to channel numbers may refer to absolute RDOS/DOS channels or to program relative BASIC channels. A negative channel number refers to the RDOS/DOS channel (e.g., -6 refers to RDOS/DOS channel 6). A positive channel number refers to a file opened in the current BASIC program on this relative channel (e.g., 2 refers to the file OPENed in the current program on channel 2). Therefore, channel numbers for STMC's may range from -63 to 15. RDOS/DOS channel 0 is a special case that is passed as -0 (-32768). You must be very careful when using absolute channel numbers.

The general format for STMC is:

STMC call, error, arguments

where call is one of the following calls (by number), error is the RDOS/DOS error returned (positive code returned, negate it to use with the BASIC.ER file; or -1 returned if call is successful) and arguments vary with each call.

STMC 0, error, filename, sectors

.CCONT creates a contiguous file with the name filename and the number of sectors you specified for size.

STMC 1, error, directoryname

.CDIR creates a subdirectory with the name directoryname.

STMC 2, error, attrib, channel

.CHATR changes the attributes for the file opened on channel according to the attribs you specified.

STMC 3, error, attrib, channel

.CHLAT changes the link attributes of the file opened on channel according to attribs specified.

STMC 4, error, buffer$, channel

.CHSTS returns the 18 word (36 byte) channel status table.

STMC 5, error, partitionname, sectors

.CPAR creates a subpartition with the name partitionname and the number of sectors as its size.

STMC 6, error, filename

.CRAND creates a random file with the name filename.

STMC 7, error, filename

.CREATE creates a sequential file with the name filename.

STMC 8, error, filename

.DELETE deletes the file filename.

Licensed Material-Property of Data General Corporation

STMC 9, error, directoryname

.DIR changes the current directory to the directory specified.

STMC 10, error, oldname, newname

.EQIV changes a device's old name to a new name.

STMC 11, error, code

.ERTN takes the RDOS/DOS error return, passing the error code to the previous push level.

STMC 12, error, programname, AC2value

.EXBG checkpoints the current background program and executes the program you specified. AC2 (accumulator 2) contains the priority flag and message word passed to the program to be executed.

STMC 13, error, programname, AC2value

.EXFG executes the program specified in the foreground partition. AC2 contains the priority flag and message word passed to the program to be executed.

STMC 14, error, foregroundflag

.FGND determines whether or not a forground program is already running.

STMC 15, error, inputconsole$

.GCIN returns the name of the console input device for this partition.

STMC 16, error, outputconsole$

GCOUT returns the name of the console output device for this partition.

STMC 17, error, directory$

.GDIR returns the name of the current default directory.

STMC 18, error, attrib, channel

.GTATR returns the attributes of the file opened on channel.

STMC 19, error, system$

.GSYS returns the current operating system's name.

STMC 20, error, devicename, flag

.INIT initializes a device into the system, either full or partial, that you specified in flag.

STMC 21, error, linkname, resolutionname

.LINK creates a link entry to the resolution file specified.

STMC 22, error, masterdevice$

.MDIR returns the name of the current master device.

STMC 23, error

.ODIS disables console keyboard interrupts.

STMC 24, error

.OEBL enables console keyboard interrupts.

STMC 25, error, oldname, newname

.RENAME renames a given file's old name to a new name.

STMC 26, error

.RESET closes all files currently open in this partition.

STMC 27, error, devicename

.RLSE releases a previously initialized device from the system.

STMC 28, error

.RTN returns control to the program at the previous push level.

STMC 29, error, day, month, year-1968

.SDAY sets the current system date.

STMC 30, error, devicename

.SPDA disables spooling for the device you specified.

STMC 31, error, devicename

.SPEA enables spooling for the device you specified.

STMC 32, error, devicename

.SPKL kills spooling for the device you specified.

STMC 33, error, filename, buffer$

.STAT returns the status of the file you specified.

STMAC 34, error, seconds, minutes, hours

.STOD sets the time of day in the system clock.

Licensed Material-Property of Data General Corporation

STMC 35, error, linkname

.ULNK removes the link entry with the specified name.

STMC 36, error, channel

.UPDAT updates the disk resident copy of the status of the file opened on channel.

STMC 37, error, frequency

.GHRZ returns the frequency of the system's real-time clock.

STMC 38, error, filename, mask, channel

.MTOPD opens the file you specified for direct mag tape I/O.

STMC 39, error, filename, mask, channel

.APPEND opens the file you specified for appending.

STMC 40, error, filename, mask, channel

.EOPEN opens the file you specified for exclusive use.

STMC 41, error, filename, mask, channel

.ROPEN opens the file you specified for read only access.

STMC 42, error, filename, mask, channel

.OPEN opens the file you specified for shared access.

STMC 43, error

.BREAK interrupts BASIC and saves the status in BREAK.SV.

STMC 44, error, filename, mask, channel

.TOPEN transparently opens the file specified for exclusive access, without changing the date last written or accessed.

STMC 45, error, filename, sectors, date$

.TCCONT transparently creates a contiguous file of the number of sectors you specified. The time/date status area is set to the date$ (6 bytes) you specified.

STMC 46, error, filename, date$

.TCRND transparently creates a random file. The time/date status area is set to date$ (6 bytes) you specified.

STMC 47, error, filename, date$

.TCRET transparently creates a sequential file. The time/date status area will be set to date$ (6 bytes) you specified.

STMC 48, error, AC0

.GMEM gets the current memory allocation in AC0 (accumulator 0).

STMC 49, error, AC0, AC1

.SMEM sets the current memory allocation in AC0 and AC1 (accumulators 0 and 1).

STMC 50, error, string, channel

.WRL writes a line on the channel you specified.

STMC 51, error, filename, buffer$

.RSTAT returns the status of the resolution file when a link entry is specified.

STMC 52, error, filename, sectors

.CONN creates a contiguous file with the name and size you specified without initializing the file to nulls.

STMD

STMD statements allow the system manager (AA account) to send messages to the terminal output device of any user.

The message string is sent literally, so you have to put carriage returns, control characters, etc. where needed, as ASCII decimal code numbers within angle brackets (<>). For example, <13> will cause a carriage return/line feed to occur at that position in the message string. Output of the message string stops when it encounters a null or end of string, whichever comes first.

You can specify that the system wait some time for a reply from the other terminal by including a reply$ string variable and a wait numeric variable for the time. The reply string is input according to the rules of TINPUT USING. Each character received updates the string length. If you don't get a reply in the specified amount of time, the system executes the next statement in your program without changing the contents of reply$. You can then check SYS(22) to see if a timeout occurred.

Licensed Material-Property of Data General Corporation

```
STMD 0, term, 0, message$
STMD 0, term, wait, message$, reply$
```

Neither statement sends a message to a terminal that has the no-message flag set. In both forms of the statement, term is the terminal number you want to receive the message, and message$ is a string variable or string literal in quotes containing the text of the message you are sending. In the first form, a 0 indicates no wait time and no reply. The second form includes wait -- a numeric expression or variable for the number of tenths of seconds to wait for a reply, and reply$ -- a string variable to receive the reply. TINPUT USING rules apply.

```
STMD 1, term, 0, message$
STMD 1, term, wait, message$, reply$
```

Either statement overrides any no-message flag set by term. The second form waits for a reply; the rules above apply. See Example 10-5.

Example 10-5 forces a message to the operator's terminal (1). The bell character (<7>) is included with carriage returns. Line 100 waits 30 seconds for a reply; if a timeout occurs, line 0110 directs control back to line 100.

Shutting Down Business BASIC

The KILL and QUICKILL programs reside in $SYS; only those knowing the AA (system manager) account password or the OP (system operator) password can shut down the system. Both programs ask for a password, unless you are the system manager or system operator and you signed on using the AA or OP account ID/password. You can RUN "KILL" and "QUICKILL", SWAP or CHAIN "KILL" and "QUICKILL", and execute KILL and QUICKILL from the BASIC CLI or OPCLI.

QUICKILL terminates all users immediately and forces control to return to RDOS/DOS. QUICKILL, like KILL, wipes out the current state of all users, and should only be used for emergencies.

KILL first asks you for the number of minutes left before logging off, and the message to send to all users. KILL forces the message to all users every ten seconds, decrementing a time count till log off. Each user should log off in the specified time. KILL also displays the status of all users currently using the system, showing the time each user logs off. You can interrupt the KILL program to stop shut-down procedures for any reason. When the time is up, KILL shuts down Business BASIC, returning control to RDOS/DOS (master console only). If you specify 0 number of minutes, KILL assumes a 30-second delay. If there are no users currently using the system, KILL executes a QUICKILL instead. See Example 10-6 to see how KILL works.

Example 10-5

```
.
.
0100 STMD 1, 1,300," < 13 > OP! WHAT'S HAPPENING? < 7 >< 13 > ", REPLY$
0110 IF SYS(22) THEN GOTO 0100
0120 PRINT REPLY$
.
.
.
```


Licensed Material-Property of Data General Corporation

Example 10-6.

```
*RUN "KILL" )
MINUTES TO LOG OFF: 1 )
MESSAGE: GO HOME! )


00   I    D    R   13   USER     EDIT       ABFPG   02   0    8914    88.2   5025   15:45
02   I    D    C   13   MYDIR    SCRATCH    ABALB   03   0    10228   113.9  1462   15:49
```

At terminals 2 and 3, this message would occur:

PLEASE LOG OFF: 50 SECONDS TO LOG OFF

GO HOME!

PLEASE LOG OFF: 40 SECONDS TO LOG OFF

GO HOME!

When the countdown completes, this message appears at the master console for RDOS/DOS, and the R prompt for for the operating system appears:

BUSINESS BASIC IS DOWN, RDOS/DOS STILL RUNNING

R

Licensed Material-Property of Data General Corporation

User Status Table

The User Status Table contains information related to the job. There is one such table for each job. You can retrieve the address of the user status table for the current job with an STMB 1,1,14,A. You should make all changes to this table using STMAs. Some programs (HELLO, STAT, KILL, OPCLI) may require direct access to this information. If you modify data in the user tables you may cause a system crash. If you assemble the file USER.SR, it will provide the relative location of this data in the user table. Table 10-2 shows the user status table values.

Table 10-2. User Status Table Values

U.S	Status word one (Bit flags as follows)
	32768 BYE in progress (Set by BYE command before SWAP to HELLO).
	16384 Reset ON IKEY flag (Set by STMB 8).
	8192 Job is signed on (Set by HELLO).
	4096 Time slice has expired, OK to swap job.
	2048 Job is awaiting a LOCK request.
	1024 Job has loaded an overlay.
	512 Job is in queue to be swapped in.
	256 Job is in queue to be swapped out.
	128 Job was just swapped in.
	64 Job was just swapped out.
	32 Do not swap this job out at this time.
	16 Job is in queue to be mapped into address space.
	8 Job is waiting on an ESC to log it on.
	4 Job is not mapped into address space.
	2 Job is waiting on terminal input.
	1 Job is waiting on terminal output.
U.S2	Status word two (Bit flags as follows):
	32768 Secondary IKEY or unpend received (SYS(10)).
	16384 (Unused)
	8192 (Unused)
	4096 User is allowed to run START program.
	2048 User is allowed to run CLI program.
	1024 Job does not want to be IKEY'ed at this time.
	512 User is allowed CHATR and CHLAT commands in CLI.
	256 User may change directories.
	128 Filenames may have colons in open statements.
	64 User does not want to receive messages.
	32 Job will be BYE'ed off if input times out.
	16 Column counter is disabled.
	8 (Unused).
	4 Allow control characters in input.
	2 Allow lowercase characters in input and LIST.
	1 Don't echo input.
U.REC	Word for .XMT/.REC.
U.CPU	CPU time used in clock ticks (double integer).
U.IOU	Number of .SYSTEM calls made (double integer).
U.LOG	Log on time (minutes past midnight).
U.TCB	Core address of TCB (task control block).
U.JBN	Job number.

Licensed Material-Property of Data General Corporation

Table 10-2. User Status Table Values (continued)

U.TLN	Terminal line number currently attached to this job.
U.DS	Single integer length of current directory, followed by the current directory (up to 10 bytes).
U.ACN	Account number (6 bytes).
U.PNM	Program name (14 bytes).
U.PSL	Current push level.
U.P	Core address of program segment.
U.PL	Length of program segment.
U.D	Core address of data segment.
U.DL	Length of data segment.
U.CMX	Maximum core allowed for this job.
U.PRI	I/O Priority (left byte) Compute priority (right byte).
U.SPR	System call priority (right byte).
U.ESC	Bit 32768 set if IKEY has occurred. Remaining bits are an address to jump to when program tests the first bit.
U.PR	Processing mode: 0 not logged on 1 compiling 2 running 3 both (i.e., ENTER or ERASE)
U.DFO	Default output queue (6 bytes).
U.DEL	Delete character echo (left) delete character (right). 32768 Terminal is CRT (send left byte, space, left byte).
U.CAN	De-attach character (left) Line cancel character (right).
U.DLS	Characters echo on delete line.
U.PND	Primary and secondary unpend characters. 65535 Unpend on every character.
U.ESD	Primary and secondary IKEY characters.
U.TBN	Current byte count.
U.TBC	Current byte pointer.
U.TBS	Pointer to start of I/O buffer.
U.ACO	Place to save AC0 for terminal I/O.

Licensed Material-Property of Data General Corporation

Table 10-2. User Status Table Values (continued)

U.MLN	Maximum number of characters to input. Negative number is to unpend when full.
U.TPW	Page width.
U.TTS	Tab size.
U.TCC	Output character counter.
U.PAD	Pad count (left); pad character (right).
U.PCN	Pad counter (for termainal handler).
U.TYP	Terminal type.
U.ERC	Error code.
U.USR	User words (2).
U.RAD	Restart address (for terminal handler).
U.IRC	Input ring control (6 words). Current count Maximum size Input pointer Output pointer Beginning pointer End pointer
U.ORC	Output ring control (6 words as above).
U.ACT	Output active flag (waiting for output done interupt).
U.SVA	Address of extended task save area.
U.TRS	Address of TR save stack.
U.WDS	Number of words to allocate upon swap in.
U.EDA	Extended memory data address.
U.EPA	Extended memory program address.
U.SCH	System control number (ENTER, SAVE, LOAD, etc.).
U.UCH	User channel table (mode (left), RDOS channel number (right). 32768 INFOS pseudo channel (remaining bits) 16384 End of file (for non-INFOS files).
U.IOB	I/O buffer area (134 bytes).

Remainder is stack and extended save areas.

End of Chapter

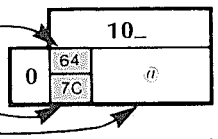
Licensed Material-Property of Data General Corporation

Appendix A
ASCII Chart

To find the *octal* value of a character, locate the character, and combine the first two digits at the top of the character's column with the third digit in the far left column.

LEGEND:

Character code in decimal
EBCDIC equivalent hexadecimal code
Character



OCTAL	00_		01_		02_		03_		04_		05_		06_		07_	
0	0 00	NUL	8 16	BS (BACK-SPACE)	16 10	DLE ↑P	24 18	CAN ↑X	32 40	SPACE	40 4D	(	48 F0	0	56 F8	8
1	1 01	SOH ↑A	9 05	HT (TAB)	17 11	DC1 ↑Q	25 19	EM ↑Y	33 5A	!	41 5D	)	49 F1	1	57 F9	9
2	2 02	STX ↑B	10 15	NL (NEW LINE)	18 12	DC2 ↑R	26 3F	SUB ↑Z	34 7F	" (QUOTE)	42 5C	*	50 F2	2	58 7A	:
3	3 03	ETX ↑C	11 0B	VT (VERT. TAB)	19 13	DC3 ↑S	27 27	ESC (ESCAPE)	35 7B	#	43 4E	+	51 F3	3	59 5E	;
4	4 37	EOT ↑D	12 06	FF (FORM FEED)	20 3C	DC4 ↑T	28 1C	FS ↑\	36 5B	$	44 6B	, (COMMA)	52 F4	4	60 4C	<
5	5 2D	ENQ ↑E	13 0D	RT (RETURN)	21 3D	NAK ↑U	29 1D	CS ↑]	37 6C	%	45 60	-	53 F5	5	61 7E	=
6	6 2E	ACK ↑F	14 0E	SO ↑N	22 32	SYN ↑V	30 1E	RS ↑↑	38 50	&	46 4B	. (PERIOD)	54 F6	6	62 6E	>
7	7 2F	BEL ↑G	15 0F	SI ↑O	23 26	ETB ↑W	31 1F	US ↑←	39 7D	' (APOS)	47 61	/	55 F7	7	63 6F	?

OCTAL	10_		11_		12_		13_		14_		15_		16_		17_	
0	64 7C	@	72 C8	H	80 D7	P	88 E7	X	96 79	` (GRAVE)	104 88	h	112 97	p	120 A7	x
1	65 C1	A	73 C9	I	81 D8	Q	89 E8	Y	97 81	a	105 89	i	113 98	q	121 A8	y
2	66 C2	B	74 D1	J	82 D9	R	90 E9	Z	98 82	b	106 91	j	114 99	r	122 A9	z
3	67 C3	C	75 D2	K	83 E2	S	91 8D	[	99 83	c	107 92	k	115 A2	s	123 C0	{
4	68 C4	D	76 D3	L	84 E3	T	92 E0	\	100 84	d	108 93	l	116 A3	t	124 4F	|
5	69 65	E	77 D4	M	85 E4	U	93 9D	]	101 85	e	109 94	m	117 A4	u	125 D0	}
6	70 C6	F	78 D5	N	86 E5	V	94 5F	↑ or ^	102 86	f	110 95	n	118 A5	v	126 A1	~ (TILDE)
7	71 C7	G	79 D6	O	87 E6	W	95 6D	← or _	103 87	g	111 96	o	119 A6	w	127 07	DEL (RUBOUT)

SD-00217 Character code in octal at top and left of charts.

↑ means CONTROL

End of Appendix

Licensed Material-Property of Data General Corporation

Appendix B
Keyword Summary

This appendix contains a summary of all keywords used in Business BASIC. Each keyword has a format, description, type, and chapter reference. Statements must be program statements with line numbers; commands are keyboard mode commands; you use functions and expressions in statements; you must enter subroutines before you GOSUB to their entry points; and you can RUN, CHAIN, or SWAP to utility programs. Use BASIC CLI commands with the BASIC CLI; use FM, SM, EDIT, and DOC commands and functions with their respective utilities.

Keyword/Format and Description	Type	Chapter
@(line, column) Cursor positioning expression for PRINT and INPUT.	Expression	3
@(item,value) Terminal characteristics expression for PRINT and INPUT, to set a value for item. Items are: -1 line-cancel key -2 character-delete echo -3 character-delete key -4 primary unpend key -5 secondary unpend key -6 primary interrupt key -7 secondary interrupt key -8 page width -9 tab size -10 maximum characters allowed on input -12 first-echoed character for line-cancel key -13 second-echoed character for line-cancel key -14 pad character -15 number of pad characters to send -19 reset items -1 to -15 to terminal default values	Expression	3
@(item) Terminal control expression for PRINT and INPUT to perform function indicated by item. Items are: -20 move cursor to *home* -21 move cursor right one column -22 move cursor down one line -23 move cursor left one column -24 move cursor up one line -25 sound the bell or audible alarm on terminal -26 tab to next field on screen -27 return cursor to first column of current line -28 move cursor to beginning of next line -30 clear entire screen (or generate form feed) -32 clear to end of line -38 start low intensity field (6053 terminals only) -39 start high intensity field (6053 terminals only) -40 start blinking field -41 end blinking field -52 start underscored field -53 end underscored field	Expression	3

Licensed Material-Property of Data General Corporation

Keyword/Format and Description	Type	Chapter
ABS (expr) Absolute value of expr.	Function	3
AND ($expr_1$, $expr_2$) Logical binary AND of $expr_1$ and $expr_2$.	Function	3
APPEND appendfile $oldfile_1$ *[$old file_2$ ...]* Append file(s) to appendfile.	CLI command	8
ASC (strvar) Extract a number from strvar.	Function	3
ASG devicename Assign a device to current job for exclusive use.	CLI Command	8
ATTACH Attach your terminal to a de-attached job.	Utility Program	7
BLDCOM $filename_1$ *[$filename_2$ ...]* Build a documentation file for PRTCOM.	CLI Command	8
BLOCK READ { strvar / var } Read the common area (512 bytes).	Statement and Command	3(4)
BLOCK READ FILE (chan), { strvar / var } Read blocks (512 byte sectors) from file.	Statement and Command	3(4)
BLOCK WRITE { strvar / var } Write a 512-byte string or array to the common area.	Statement and Command	3(4)
BLOCK WRITE FILE (chan), { strvar / var } Write 512 byte blocks to a file.	Statement and Command	3(4)
BUILD outputfile $filename_1$ *[$filename_2$ ...]* Build a file of filenames.	CLI Command	8

Licensed Material-Property of Data General Corporation

Keyword/Format and Description	Type	Chapter
BYE Log off the system.	Statement and Command	3
CCONT filename$_1$ blocks$_1$ *[filename$_2$ blocks$_2$ ...]* Create a contiguous file.	CLI Command	8
CDIR (sub)directoryname Create a subdirectory or directory.	CLI Command	8
CHAIN filename Execute a program (overwrite CLI).	CLI Command	8
CHAIN { strvar / "filename" } *[THEN* { *GOTO linenum* / *CON* } *]* Execute a utility or another program.	Statement and Command	3
CHATR filename$_1$ attrib$_1$ *[filename$_2$ attrib$_2$ ...]* Change a file's attributes.	CLI Command	8
CHLAT filename$_1$ attrib$_1$ *[filename$_2$ attrib$_2$ ...]* Change a file's link access attributes.	CLI Command	8
CHR$(expr *[,byte])* Put binary value of a number into a string.	Function	3
CLI Execute the BASIC CLI utility program.	Utility Program	7
CLOSE *[FILE(chan)]* Close all open files or a specific file.	Statement and Command	3
CON Continue execution of a stopped program.	Command	3
CPART partname blockcount Create a secondary partition.	CLI Command	8
CRAND filename$_1$ *[filename$_2$ ...]* Create a random file.	CLI Command	8
CREATE filename$_1$ *[filename$_2$ ...]* Create a sequential file (RDOS only).	CLI Command	8

Licensed Material-Property of Data General Corporation

Keyword/Format and Description	Type	Chapter
CRM$ (strvar) Cram three bytes of a string into two bytes.	Function	3
DATA value *[,value ...]* Data for variables in READ statements.	Statement	3
DEF FNa(d)=expr Define your own function.	Statement	3
DELAY expr Delay execution of the next program statement or command.	Statement and Command	3
DELETE filename$_1$ *[filename$_2$ ...]* Delete a file, directory, or partition.	CLI Command	8
DELETE { strvar / "filename" } Delete a file from your directory.	Statement and Command	3
DELREC.SL Delete a record and add it to deleted record chain. Line numbers: 8600-8695 Entry point: 8600	Subroutine	7(4)
DIM { svar$(n) / array(col) / array(row,col) } [, { *svar$(n)* / *array(col)* / *array(row,col)* } ...] Dimension arrays and/or strings.	Statement and Command	3
DIR { partname:directory / directory } Change the current directory.	CLI Command	8
DISK Display the amount of disk space that was used and what is left.	CLI Command	8
DIV.SL Calculate extended percentage. Line numbers: 7550-7590 Entry point: 7550	Subroutine	7

Licensed Material-Property of Data General Corporation

Keyword/Format and Description	Type	Chapter
DOC Print a series of text input files. DOC has a series of control commands to be included in text to control output: \CPAGE *[n]* Conditional page-up. \ECPAGE End conditional page-up. \EQC Turn off centering. \ETX Turn off justification. \HEAD n, text Print section/page heading. \MAR l,r Set left and right margins. \NLF No line feed. \PAGE *[p,l]* Page-up with optional page settings. \QC *[n]* Start centering lines of text. \SPAGE n Set the current page number. \STX Turn on justification.	Utility	7
DOCTOC Print a table of contents.	Utility	7
DUMP outputfile *[filename$_1$ filename$_2$ ...]* Copy diskfiles to an output file.	CLI Command	8

Licensed Material-Property of Data General Corporation

Keyword/Format and Description	Type	Chapter
EDIT Create and edit text files. **Insert Mode:** ikey (interrupt key) Enter and exit insert mode. **Comment Mode:** COM Enter comment mode. ikey (interrupt key) Exit comment mode. **Command Mode:** *[+/-]*line Go to a line, or forwards/backwards a number of lines. *[+/-]*line *[:]*Text Replace (:) or insert text at line or before (-) or after (+) a line. A *[:][n]* Append lines or page to edit buffer from input file or alternate (:) input file. ACC Allow control characters. ALC Allow lowercase characters. B Go to bottom of edit buffer and insert mode. BR Brief mode. C/string$_1$ /string$_2$ $\left[\left\{ \begin{array}{l} /A \\ /G\left\{ n_1, n_2 \atop +/- \right\} \end{array} \right\}\right]$ Change first occurence or all (G) occurences of string$_1$ to string$_2$ in the current line, for + or - lines from the current line, or lines n_1 to n_2, or all occurrences in all *(A)* lines in the edit buffer.	Utility	7

Licensed Material-Property of Data General Corporation

Keyword/Format and Description	Type	Chapter
COM Enter comment mode.		
COPY *[:]* { string / *n* } Copy n lines or until string is found from input file or alternate (:) input file to output file.		
D { +/- *n* / *line*$_1$, *line*$_2$ } Delete the current line, or + or - *n* lines from the current line, or *line*$_1$ to *line*$_2$ in edit buffer.		
DCC Disallow control characters.		
DITTO line$_1$, line$_2$ Copy line$_1$ to line$_2$ to lines following the current line (does not delete copied lines).		
DLC Disallow lowercase characters.		
END Output buffer, copy remaining input file to outputfile, and rename outputfile to inputfile (deleting inputfile).		
F *[:]*filename Specify input file or alternate (:) input file.		
FIND string Find all occurrences of string from current line to end of buffer.		
I string Insert string after current line in edit buffer.		
IB string Insert string before current line in edit buffer.		
L string Locate next occurrence of string in edit buffer.		
LC string Locate all remaining occurrences of string in edit buffer.		
LINES n Set the number of lines (n) for a page (default is 60).		

Licensed Material-Property of Data General Corporation

Keyword/Format and Description	Type	Chapter
EDIT (continued) MAX n Set the maximum line length n (default is 132).		
MOVE $line_1$, $line_2$ Move $line_1$ to $line_2$ to lines following the current line, and delete $line_1$ to $line_2$.		
N Go to and display the next line in the edit buffer.		
O $\left[\left\{\begin{matrix} +/- n \\ line_1, line_2 \end{matrix}\right\}\right]$ Output *n* lines, or + or - *n* lines from the current line, or $line_1$ to $line_2$, or entire edit buffer to output file.		
OMIT Stop EDIT, leave all temporary files open and unchanged.		
OW $\left[\left\{\begin{matrix} [+/-]n \\ line_1, line_2 \end{matrix}\right\}\right]$ Same as O, but puts a page mark on the last line.		
P $\left\{\begin{matrix} [+/-]n \\ line_1, line_2 \end{matrix}\right\}$ Print (display) n lines, or for + or - *n* lines from the current line, or $line_1$ to $line_2$.		
P! Print (display) all lines in the edit buffer.		
PAGE *[:]* Output buffer and bring in next page from input file or alternate (:) input file.		
R string Replace the current line with string.		
SET loc_1 *[*,loc_2 ... loc_8 *]* Set tab locations on a line (up to 8).		
SKIP *[:]* $\left\{\begin{matrix} n \\ string \end{matrix}\right\}$ Skip n lines or until string is found in input file or alternate (:) input file and discard lines.		

Licensed Material-Property of Data General Corporation

Keyword/Format and Description	Type	Chapter
STOP Execute an END and terminate the EDIT utility.		
T Go to top (first line) of edit buffer.		
V Enter verify mode, always on unless suppressed by BR.		
END Optional program terminator.	Statement	3
ENTER { "filename" / strvar } Bring in to working storage program source statements.	Statement and Command	3
EOF (chan) Test for end of file.	Function	3
EQUIV newname oldname Rename a device (RDOS only).	CLI Command	8
ERASE $line_1$, $line_2$ Erase program statements.	Statement and Command	3
ERM$({ num / code }) Get an error message.	Function	3
FDUMP magtapefile *[filename$_1$ filename$_2$ ...]* Fast dump of files to mag tape (RDOS only).	CLI Command	8
FILCOM $filename_1$ $filename_2$ *[filename/L]* Compare two files, output to *filename/L*.	CLI Command	8
FILES Display names of files in current directory.	CLI Command	8
FILESORT Sort a data file.	Utility	7
FILL$ (ascii *[,bytes]*) Fill a string with an ASCII value, or fill a specified number of bytes.	Function	3

Licensed Material-Property of Data General Corporation

Keyword/Format and Description	Type	Chapter
FINDFILE.SL Find a subfile and build-C1 array. Line numbers: 7800-7845 Entry point: 7800	Subroutine	7(4)
FLOAD magtapefile *[filename$_1$ filename$_2$ ...]* Fast load FDUMPed files.	CLI Command	8
FM File Maintenance utility package. File Maintenance has the following functions, and F *n* keys are for DG 6053 terminals: ADD (F4 key) Add a new record. AUTO Auto display (roll-mode FM only). CHANGE (F3 key) Change a record. carriage return) or tab key Move to next field. DELETE (F6 key) Delete a record. DISPLAY Display a record (roll-mode FM only). home key Complete function (FM only). F11 key Clear an error (FM only). F12 key or backtab key. Previous field (FM only). FIND (F1 key) Find a record. FORMAT (F5 key) Change format.	Utility	5

Licensed Material-Property of Data General Corporation

Keyword/Format and Description	Type	Chapter
File Maintenance (continued) NEXT (F2 key) Find next record. PAGE (F7 key) Change page. STOP (F10 key) Stop FM.		
FMLOG Print a log file associated with FM.	Utility	5
FMPRINT Print an FM data file.	Utility	5
FMTABPRINT Print an FM table file.	Utility	5
FOR ctrlvar = $expr_1$ TO $expr_2$ *[STEP $expr_3$]* . . .	Statement	3
NEXT ctrlvar Define a program loop.	Statement	3
FORM.SL Formatted screen input/output. Line numbers: You need UNFORM.SL (9100-9895) and VAL.SL (7600-7659). Entry points: 9300 Input string or numeric field. 9500 Output string or numeric field.	Subroutine	7
FPRINT filename Dump a file to the line printer or terminal.	CLI Command	8
FREE filename Free a device from exclusive (ASG) use.	CLI Command	8
GDIR Display the current directory name.	CLI Command	8

Licensed Material-Property of Data General Corporation

Keyword/Format and Description	Type	Chapter
GETCM.SL Read the CLI COM file. Line numbers: 7500-7560 Entry points: 7500 Read command line. 7550 Initialize variables and read common area.	Subroutine	7
GETREC.SL Get next available record. Line numbers: You need POSFL.SL (9610-9645) with GETREC.SL (8400-8495). Entry point: 8400	Subroutine	7(4)
GOSUB line line . . . . . (subroutine) . RETURN Transfer to and from a subroutine within a program.	Statements	3
GOTO line Go to a specific statement.	Statement	3
GPOS (chan) Current byte position in a file.	Function	3
GSDIR Display current system directory name.	CLI Command	8
GSYS Display the current system name.	CLI Commmand	8
GTOD Display the time and date.	CLI Command	8
IBUILD Build an index file from sorted tag or data file.	Utility	7(4)
IF { relexpr / expr } { THEN statement / { THEN / GOTO / GOSUB } line } Transfer to or execute a statement if a condition is true.	Statement	3

Licensed Material-Property of Data General Corporation

Keyword/Format and Description	Type	Chapter
INDEXBLD Build an index file.	Utility	7(4)
INDEXCALC Calculate index information.	Utility	7
INDEXPRT Print an index file.	Utility	7
INIT devicename Initialize a device.	CLI Command	8
INITFILE Initialize an index or data file.	Utility	7(4)
INITINDEX.SL Initialize an index file. Line numbers: 7700-7790 Entry point: 7700	Subroutine	7
INPUT [{ *FILE (chan)* / *[@ (num),] "prompt,"* }] [*var [,var...]*] *[;]* Input data in character format from a terminal or from a file.	Statement and Command	3
INPUT [{ *FILE (chan), USING"",* / *USING"", [@ (num), "prompt,"]* }] var *[,var...][strvar]* Input characters from a terminal or file, allowing error and terminator trap.	Statement and Command	3
INIT (expr) Truncate a number to make it an integer.	Function	3
KADD desc$,buff$,key$,recno Add a key to an index file.	Statement and Command	3(4)
KDEL desc$,buff$,key$,recno Delete a key from an index file.	Statement and Command	3(4)
KFIND desc$,buff$,key$,recno Find a key in an index file.	Statement and Command	3(4)
KNEXT desc$,buff$,key$,recno Find the next key in sequence after KFIND.	Statement and Command	3(4)

Licensed Material-Property of Data General Corporation

Keyword/Format and Description	Type	Chapter
LEN(strvar) Find the current length of a string.	Function	3
[LET] var = expr Assign a value to a variable.	Statement and Command	3
LIBRARY Display names of files in the library directory.	CLI Command	8
LINK linkname filename Link an alternate name to a file.	CLI Command	8
LIST filename$_1$ *[filename$_2$ ...]* List information for files in the current directory.	CLI Command	8
LIST $\left[\left\{\begin{array}{l} line \\ TO\ line \\ line_1\ ,\ \left\{\begin{array}{c} TO \\ , \end{array}\right\}\ line_2 \end{array}\right\}\right]$ $\left[\text{"}filename\text{"}\right]$ List the character contents of a program to a file or a terminal.	Command	3
LOAD inputfile *[filename$_1$ filename$_2$ ...]* Reload dumped files.	CLI Command	8
LOAD "filename" Load a SAVEd program.	Command	3

Licensed Material-Property of Data General Corporation

Keyword/Format and Description	Type	Chapter
LOCK iden, {"filename" / strvar} start,recsize *[,time]* UNLOCK *[iden]* File lock/unlock device.	Statement and Command Statement and Command	3(4) 3(4)
LOCKS Display your current locks.	Utility	7
LSPEED Set your terminal's speed.	Utility	7
LISTCOM listfile$_1$ listfile$_2$ *[outputfile]* Compare two listing (ASCII format) files.	CLI Command	8
LSTMERGE listfile commentedfile *[outputfile]* Merge two listing (ASCII format) files.	CLI Command	8
MAX(expr$_1$, expr$_2$) Find maximum of two expressions.	Function	3
MDIR Display the master directory name.	CLI Command	8
MIN(expr$_1$, expr$_2$) Find the minimum of two expressions.	Function	3
MOD(expr$_1$, expr$_2$) Find the remainder after dividing two expressions.	Function	3
MOVE directory *[filename$_1$ filename$_2$ ...]* Move files from one directory to another.	CLI Command	8
MOVETABREC Copy table file records (FM and roll-mode FM).	Utility	5
MSG, *[terminal,]* message Send a message to another terminal.	Command	3

Licensed Material-Property of Data General Corporation

Keyword/Format and Description	Type	Chapter
MTDIO chan, func { strvar, er / records, er / er } Direct I/O control of mag tape and cassette. func: 1 read a tape record 2 rewind tape 4 skip records forward 5 skip records backward 6 write a tape record 7 write an end-of-file mark 8 read tape drive status	Statement and Command	3
MUL.SL Calculate percent of a number. Entry point: 7500	Subroutine	7
NEW Clear your working storage area.	Statement and Command	3
NEXT (See FOR/NEXT.)		
NFOS CLOSE E, *[C,B$]* Close an INFOS file.	INFOS statement	9
NFOS DELETE E, *[C,B$,L]*;posspec Delete an INFOS record.	INFOS Statement	9
NFOS DELSI E, *[C,B$]*;posspec Delete a subindex.	INFOS Statement	9
NFOS DFSI E, *[C,B$,L]*,KL,PL,NS;posspec Define a subindex.	INFOS Statement	9
NFOS LNKSI E, *[C,B$,L]*;posspec TO posspec Link to an existing subindex.	INFOS Statement	9
NFOS OPEN E, *[C,B$]*,filename Open an INFOS file.	INFOS Statement	9
NFOS READ E, *[C,B$,L]*,D$,P$;posspec Read a record or partial record.	INFOS Statement	9
NFOS REINS E, *[C,B$,L]*;posspec Reinstate a logically deleted record.	INFOS Statement	9

Licensed Material-Property of Data General Corporation

Keyword/Format and Description	Type	Chapter
NFOS RETDF E, *[C,B$]*,KL,PL,NS;posspec Retrieve a subindex definition.	INFOS Statement	9
NFOS RETHK E, *[C,B$]*,K$;posspec Retrieve the highest key in an index.	INFOS Statement	9
NFOS RETKY E, *[C,B$]*,K$;posspec Retrieve a key in an index.	INFOS Statement	9
NFOS RETST E, *[C,B$]*,posspec Retrieve record status.	INFOS Statement	9
NFOS REWRITE E, *[C,B$,L]*,D$,P$;posspec Rewrite a record.	INFOS Statement	9
NFOS WRITE E, *[C,B$,L]*,D$,P$;posspec Write a record.	INFOS Statement	9
ON ERR THEN statement Trap an error in your program.	Statement	3
ON expr GOSUB line$_1$ *[,line$_2$ ...]* Conditional transfer to subroutine(s).	Statement	3
ON expr {GOTO / THEN} line$_1$ *[line$_2$ ...]* Conditional transfer to statement(s).	Statement	3
ON IKEY THEN statement Trap an interrupt in your program.	Statement	3
OPEN Open physical files (for subfiles).	Utility	7(4)
OPEN FILE (chan,mode), {"filename" / strvar} Open a file or device in an access mode and give it a channel number.	Statement and Command	3(4)

Licensed Material-Property of Data General Corporation

Keyword/Format and Description	Type	Chapter
OPEN FILE (continued) Modes: 0 random exclusive I/O, create 1 sequential exclusive output, create 2 sequential exclusive output, append 3 sequential shared input 4 random shared input 5 random shared I/O 6 random exclusive I/O 7 MTDIO (mag tape) I/O		
OR(expr$_1$, expr$_2$) Inclusive logical OR of two expressions.	Function	3
PAGE = width Set the page width (default 72, max 132).	Command	3
POP Exit the BASIC CLI and clear common area.	CLI Command	8
PORTS Display jobs on the system.	Utility	7
POSFL.SL Position a file pointer to a record. Line numbers: 9610-9645 Entry points: 9610 Beginning of record. 9612 Byte offset within record.	Subroutine	7(4)
POSITION FILE (chan,byte) Position file pointer to a byte in a disk file.	Statement and Command	3(4)
PRINT filename$_1$ *[filename$_2$ ...]* Print a file on the line printer.	CLI Command	8
{ PRINT / ; } *[FILE(chan),] [@ (num);] [TAB(n);] [expr] [{ ; / , } expr...]* Output characters to terminal, output file or device.	Statement and Command	3
{ PRINT / ; } *[FILE(chan),]* USING { "format" / strvar }, expr *[expr...]* PRINT *[FILE]* using formats.	Statement and Command	3

Licensed Material-Property of Data General Corporation

Keyword/Format and Description	Type	Chapter
PROTFORM.SL Protect/unprotect a formatted screen field. Line numbers: You need UNFORM.SL (9100-9895) and VAL.SL (7600-7659). Entry points: 9200 Converts to unprotect and output field. 9250 Convert to protect and output field.	Subroutine	7
PRTCOM filename Print a BLDCOM documentation file.	CLI Command	8
QFILESORT Fast sort (ascending) of a file.	Utility	7
QUIT Exit the BASIC CLI (retains common area).	CLI Command	8
RANDOMIZE Start/reseed random number generator.	Statement and Command	3
READ var$_1$ *[, var$_2$ ...]* Get values in DATA for variables.	Statement	3
READ FILE (chan), var *[,var ...]* Input binary data from a file.	Statement and Command	3(4)
RELEASE { unit / directory } Release a device or directory.	CLI Command	8
RELINK Clean up a linked data file.	Utility	7
REM text A remark statement.	Statement	3
RENAME oldname$_1$ newname$_1$ *[oldname$_2$ newname$_2$ ...]* Rename files.	CLI Command	8
RENAME { "oldname" / strvar } , { "newname" / strvar } Rename directory entries for files.	Statement and Command	3

Licensed Material-Property of Data General Corporation

Keyword/Format and Description	Type	Chapter
RENUMBER { line / STEP increment / line STEP increment } Renumber lines in the current program.	Command	3
REPLACE { *"filename"* / *strvar* } Replace a program on disk with a new program.	Statement and Command	3
RESTORE *[line]* Reset DATA data list pointer.	Statement and Command	3
RETURN (See GOSUB/RETURN.)		
RNAM Rename program variables.	Utility	7
RND(expr) Produce a pseudo-random integer.	Function	3
RUN [{ *"filename"* / *line* }] Execute a program.	Command	3
SAVE { "filename" / strvar } Save a program on disk.	Statement and Command	3
SCHANS Display the system channel assignments.	Utility	7
SDIR directory Set the system directory (RDOS only).	CLI Command	8
SGN(expr) Determine the sign of an expression.	Function	3
SHFT(expr,bits) Logical bit shift of a value's bits.	Function	3
SIZE Determine the size of the current program.	Command	3

Licensed Material-Property of Data General Corporation

Keyword/Format and Description	Type	Chapter
SLINE port/L *[width/W]* Select a line (terminal) and attach.	CLI Command	8
SM Screen Maintenance utility package. SM has the following commands and functions and F *n* keys on 6053 terminals: DELETE CHAR (Shift-F2) Delete a character DELETE LINE (Shift-F4) Delete a line. DISPLAY *[screen]* Command to display current or specified screen. DUP TO EOL (F1) Duplicate previoius line to end of line. END PROTECT (F5) End protection, start unprotected field. ENTER CMND (F4) Position cursor to command line to enter a command. INSERT CHAR (Shift-F1) Create a space to insert a character. INSERT LINE (Shift-F3) Create a space to insert a line. PRINT *[filename]* Command to print current screen to file or line printer. PROCESS CMND Execute the command typed at the command line. PROTECT TO EOL (F2) Protect to end of line. READ *[screen]* Command to read current or specified screen. SCALE TO EOL (F3) Fill line with number scale.	Utility	6

Licensed Material-Property of Data General Corporation

Keyword/Format and Description	Type	Chapter
SM (continued) START PROTECT (F6) End unprotected field, start protection. STOP Stop an SM session. TAB TO NEXT (F9) Move cursor to next unprotected field. VERIFY SCREEN (F10) Redisplay the current screen. WRITE *[screen] [,system,project,program]* Command to output current screen to screen file.		
SPDIS device$_1$ *[device$_2$ ...]* Disable device spooling (RDOS only).	CLI Command	8
SPEBL device$_1$ *[device$_2$ ...]* Enable device spooling (RDOS only).	CLI Command	8
SPKILL device$_1$ *[device$_2$ ...]* Delete the spool queue (RDOS only).	CLI Command	8
SQR(expr) Square root (truncated to integer) of expr.	Function	3
START *[prog/N] [accnt/A] [passw/P] [dir/D] ["input$_1$ "] ["input$_2$ "...]* Start a de-attached job.	CLI Command	8
STAT Display the status of all jobs.	CLI Command	8

Licensed Material-Property of Data General Corporation

Keyword/Format and Description	Type	Chapter
STMA type, arguments System Calls. The types of system calls are: STMA 1,item,var Receive the value of item. STMA 2,item,value Set the value of item. Items are: 0 terminal type 1 error code passed between programs 2 available to pass data (1 word) 3 available to pass data (1 word) STMA 3,item,var Receive value of item. STMA 4,item,value Set the value of item. Items are: 0 de-attach key 1 line-cancel key 2 character-delete echo 3 character-delete key 4 primary unpend key 5 secondary unpend key 6 primary interrupt key 7 secondary interrupt key 8 page width 9 tab size 10 maximum characters allowed on input 12 first-echoed character for line-cancel key 13 second-echoed character for line-cancel key 14 pad character 15 number of pad characters to send STMA 5,flag,var Retrieves the value of flag. STMA 6,flag Sets flag.	Statement and Command	3

Licensed Material-Property of Data General Corporation

Keyword/Format and Description	Type	Chapter
STMA (continued) STMA 7,flag Clears flag. Flags are: 0 no echo 1 allow lowercase characters to be input 2 allow control characters to be input 3 disable the column counter 4 no messages allowed 5 no IKEY 6 suppress listing of GOTO/GOSUB lables STMA 8,0 Reset FOR/NEXT and GOSUB/RETURN stacks. STMA 8,1 Reset only the FOR/NEXT stack. STMA 8,2 Reset only the GOSUB/RETURN stack. STMA 8,3 Reset SYS(26) IKEY indicator. STMA 8,4 Flush the input buffer. STMA 9,item,strvar Receives the value of item. Items are: 0 account specifier 1 current directory name 2 current program name 3 current default output device STMA 10,0,string Set special characters allowed in crammed (CRM$/UCM$) string conversions. STMA 11,julian,var_1, var_2, var_3 Convert a julian date to month, day, and year. STMA 12,var,month,day,year Convert a date into a julian date.		

Licensed Material-Property of Data General Corporation

Keyword/Format and Description	Type	Chapter
STMA (continued) STMA 13,strvar,tablestring Translate a string of characters according to your own table. STMA 14,strvar,type Translates a string of characters according to type. Types are: 0 lowercase alphabetic ASCII to uppercase only 1 ASCII code to EBCDIC code 2 EBCDIC to ASCII STMA 15,$string_1$, $string_2$, var Compare two strings. STMA 16,value De-attach the current job. Values are: 0 when job de-attaches, terminal remains inactive. 1 when job de-attaches, terminal starts log on again. STMA 17,port,error Attaches current job to a port. STMA 18,filename,var Examine, assign, or free a reserved file or device. Values of var are: 0 examine file's condition. 1 free file from assignment. 2 assign file to current job.		
STOP Stop program execution.	Statement	3
SWAP { "filename" / strvar } [THEN { *GOTO line* / *CON* }] Execute a program and return.	Statement and Command	3

Licensed Material-Property of Data General Corporation

Keyword/Format and Description	Type	Chapter
SYS(item) Return system information. Items are: 0 time of day since midnight 1 day of the month 2 month of the year 3 year in four digits 4 terminal port number 5 CPU time used 6 I/O usage (number of system calls) 7 error code of last error 8 channel number of most recent file reference 9 job number 10 alternate IKEY or unpend flag 11 global switches used when BASIC was executed 12 time of day (seconds past last minute) 13 time of day (minutes past last hour) 14 time of day (hours since midnight) 15 status word 1 (from user status table) 16 status word 2 (from user status table) 17 program length in bytes 18 data length in bytes 19 maximum program and data length 20 statement number of last error or IKEY 21 current CPU switch settings 22 time remaining after last INPUT or TINPUT 23 log on time 24 current push level 25 processor number (0=single, 1=master, 2=slave, 3=secondary single) 26 status of IKEY flag 27 time of day as hhmmss 28 date as mmddyy 29 unique number generated in a series	Function	3
TAB = width Set the tab width.	Command	3
TAB(n) Tab expression in PRINT statement.	Expression	3
TABBUILD Quick definition of arrays for FM and roll-mode FM.	Utility	5
TABLE filename$_1$ *[filename$_2$ ...]* Print a program cross-reference.	CLI Command	8
TBUILD Build a tag file.	Utility	7
TCOPY inputfile *[outputfile] [number/N]* Copy a file from tape to tape.	CLI Command	8

Licensed Material-Property of Data General Corporation

Keyword/Format and Description	Type	Chapter
TERM Change certain terminal key functions.	Utility	7
TFER inputfile outputfile recsize/R *[blocksize/B]* Copy a file between tape and disk.	CLI Command	8
TINPUT time, *[FILE (chan),] [USING "",]* var [,var ...] "prompt" Specify amount of time for INPUT.	Statement and Command	3
TPRINT Print the RDOS tuning report.	CLI Command	8
TRUN$(strvar *[,-1]*) Truncate a string.	Function	3
TYPE filename$_1$ *[filename$_2$ ...]* Display files on the terminal.	CLI Command	8
UCALL subr *[,expr]* Call an assembly language subroutine.	Statement and Command	D
UCHANS Display your channel assignments.	Utility	7
UCM$(strvar) Uncram a crammed (CRM$) string.	Function	3
UNFORM.SL Unformatted screen input/output. Line numbers: 9100-9895 Entry points: 9150 Sets necessary STMAs. 9155 Same as 9150, but no lowercase characters. 9175 Reset STMAs for normal terminal operations. 9400 Input screen field to string. 9550 Output string to screen field. 9650 Position cursor to requested field. 9700 Display screen from screen file. 9735 Reread SCRN$ but do not display. 9750 Read function key input. 9800 Convert field specification. 9850 Output an error message to screen. 9890 Clear error message from screen.	Subroutine	7
UNLINK linkname$_1$ *[linkname$_2$ ...]* Remove a link entry.	CLI Command	8

Licensed Material-Property of Data General Corporation

Keyword/Format and Description	Type	Chapter
UNLOCK (see LOCK/UNLOCK.)		
VAL.SL Convert a string to a number. Line numbers: 7600-7659 Entry points: 7600 Allow plus, minus, add decimal point. 7608 Do not allow decimal point. 7610 Unsigned integer conversion.	Subroutine	7
VAR List the variables in a program.	Utility	7
VFU *[switches]* vfu-filename Edit a format control file for a data channel line printer (RDOS only).	CLI Command	8
WRITE FILE (chan), var *[,var ...]* Output binary data to a file.	Statement and Command	3(4)
XBUILD Build an index file from a data file.	Utility	7
XFER sourcefile destinationfile. Copy one file to another file.	CLI Command	8

End of Appendix

Licensed Material-Property of Data General Corporation

Appendix C
Error Messages

This appendix lists all the error messages that Business BASIC can generate. The first number is the record number in the BASIC.ER file of error messages. The second number is the error code that BASIC returns in SYS(7). You may use either number in the ERM$ string function. You may add your own error messages at the end of the file (640 and above), using File Maintenance and table file ERRORS.TB.

Business BASIC Error Messages

1	1	ILLEGAL CHARACTER
2	2	STATEMENT OR COMMAND SYNTAX IS INVALID
3	3	READ/DATA TYPES INCONSISTENT
4	4	SYSTEM (PROGRAM IS LOST)
5	5	STATEMENT NUMBER
6	6	EXCESSIVE NUMBER OF VARIABLES (MORE THAN 348)
7	7	STATEMENT MAY NOT BE USED AS A COMMAND
8	8	SPECIFICATION
9	9	ILLEGAL RESERVED FILE NAME
10	10	RESERVED FILE IN USE
11	11	PARENTHESIS
12	12	ILLEGAL COMMAND
13	13	LINE NUMBER
14	14	NO MORE PROGRAM MEMORY
15	15	END OF DATA
16	16	ARITHMETIC
17	17	UNASSIGNED VARIABLE
18	18	GOSUB NESTING
19	19	RETURN - NO GOSUB
20	20	FOR NESTING
21	21	FOR - NO EXIT
22	22	NEXT - NO FOR
23	23	NO MORE DATA MEMORY
24	24	NO AVAILABLE CHANNELS
25	25	OPTION IS NOT IN THIS SYSTEM
26	26	PROGRAM/DATA OVERFLOW
27	27	ILLEGAL FILE NUMBER
28	28	NEW DIMENSION EXCEEDS OLD DIMENSION
29	29	EXPRESSION
30	30	ILLEGAL MODE
31	31	SUBSCRIPT
32	32	UNDEFINED FUNCTION
33	33	FUNCTION NESTING
34	34	FUNCTION ARGUMENT
35	35	ILLEGAL FORMAT STRING
36	36	STRING SIZE
37	37	USER ROUTINE
38	38	UNDIMENSIONED STRING
39	39	DUPLICATE MATRIX
40	40	MATRICES SIZES
41	41	MATRIX DIM
42	42	FILE ALREADY OPENED
43	43	MATRIX NOT SQUARE
44	44	FILE UNOPENED
45	45	ILLEGAL RECORD SIZE
46	46	INPUT INVALID

Licensed Material-Property of Data General Corporation

Business BASIC Error Messages (continued)

47	47	WRONG MODE
48	48	NOT A SAVE FILE
49	49	NO ROOM FOR DIRECTORY
50	50	INVALID OPERATOR COMMAND
51	51	USER NOT ON SYSTEM
52	52	USER IN NOMSG STATE
53	53	RENUMBERING ERROR(S)
54	54	STATEMENT LENGTH
55	55	BLOCK I/O ERROR
56	56	ATTEMPT TO LOCK SAME AREA TWICE
57	57	RECORD IN USE
58	58	INCOMPATIBLE SAVE FILE
59	59	ZERO STEP
60	60	LINE TOO LONG
61	61	MISSING RIGHT PARENTHESES IN FORMAT
62	62	SAVE FILE IS RUN ONLY
63	63	INCORRECT NUMBER OF ARGUMENTS
64	64	INCORRECT ARGUMENT TYPE
65	65	PROGRAM SWAPPING ERROR
66	66	STRING ARGUMENT IN ERROR
67	67	ERROR IN INDEX FILE OR INDEX PARAMETERS
68	68	INDEX FILE FULL
69	69	EXCESSIVE IPB ERRORS
70	70	INVALID JOB/TERMINAL NUMBER
71	71	JOB/TERMINAL ALREADY ATTACHED
72	72	JOB IS NOT WAITING ON INPUT
73	73	EDIT BUFFER IS EMPTY
75	75	PROGRAM MAY NOT BE LISTED

Utility Program Error Messages

128	128	INVALID FOR OUT-OF-RANGE FIELD
129	129	NO KEY WAS GIVEN
130	130	RECORD DOES NOT EXIST
131	131	RECORD ALREADY EXISTS
132	132	VALIDATION ERROR. KEY WAS CHANGED
133	133	FUNCTION IS INVALID WITHOUT A FIND FUNCTION FIRST
134	134	TABLE FILE IS IN ERROR
135	135	INVALID FORMAT OR PAGE
136	136	NO NEXT RECORD
137	137	INDEX FOR THIS RECORD MAY BE ALL MESSED UP
138	138	KEY COULD NOT BE DELETED
139	139	REQUEST NOT COMPLETED
140	140	INVALID ACCOUNT/PASSWORD

RDOS/DOS and I/O Error Messages

256	0	ILLEGAL CHANNEL
257	-1	ILLEGAL FILE NAME
258	-2	ILLEGAL SYSTEM COMMAND
259	-3	ILLEGAL COMMAND FOR DEVICE
260	-4	NOT A SAVED FILE
261	-5	FILE ALREADY EXISTS
262	-6	END OF FILE
263	-7	READ PROTECTED FILE
264	-8	WRITE PROTECTED FILE
265	-9	FILE ALREADY EXISTS
266	-10	FILE NOT FOUND
267	-11	PERMANENT FILE
268	-12	ATTRIBUTES PROTECTED
269	-13	FILE NOT OPENED
270	-14	SWAPPING DISK ERROR - PROGRAM LOST
273	-17	UFI IN USE

Licensed Material-Property of Data General Corporation

274	-18	LINE LIMIT
275	-19	IMAGE NOT FOUND
276	-20	PARITY
277	-21	PUSH LIMIT
278	-22	STORAGE OVERFLOW
279	-23	NO FILE SPACE
280	-24	READ ERROR
281	-25	UNIT NOT PROPERLY SELECTED
282	-26	START ADDRESS
283	-27	STORAGE PROTECT
284	-28	FILE ACCESSIBLE BY BLOCK I/O ONLY
285	-29	FILES ON DIFFERENT DIRECTORIES
286	-30	DEVICE NOT IN SYSTEM
287	-31	OVERLAY NUMBER
288	-32	FILE NOT ACCESSIBLE BY BLOCK I/O
289	-33	SET TIME
290	-34	NO MORE TCB'S
292	-36	SQUASH FILE
293	-37	DEVICE ALREADY EXISTS
294	-38	INSUFFICIENT CONTIGUOUS BLOCKS
295	-39	QTY
296	-40	TASK QUEUE TABLE
297	-41	NO ROOM FOR DIRECTORY
298	-42	ILLEGAL DIRECTORY SPECIFIER
299	-43	UNKNOWN DIRECTORY SPECIFIER
300	-44	PARTITION TOO SMALL
301	-45	DIRECTORY DEPTH EXCEEDED
302	-46	DIRECTORY IN USE
303	-47	LINK DEPTH
304	-48	FILE IN USE
305	-49	TASK ID
306	-50	COMMON SIZE
307	-51	COMMON USAGE
308	-52	FILE POSITION ERROR
309	-53	DATA CHAN MAP
310	-54	DIRECTORY NOT INITIALIZED
311	-55	NO DEFAULT DIRECTORY
312	-56	FG ALREADY ACTIVE
313	-57	PARTITION SET
314	-58	DIRECTORY IN USE
315	-59	NO ROOM FOR UFT'S
316	-60	NO DEBUG
317	-61	NOT A LINK ENTRY
318	-62	PROGRAM NOT CHECKPOINTABLE
319	-63	SYS.DR ERROR
320	-64	MAP.DR ERROR
321	-65	DEVICE TIMEOUT
322	-66	ENTRY NOT ACCESSIBLE BY A LINK
323	-67	NOT A SOURCE FILE
324	-68	RECEIVER TIMEOUT
325	-69	SYSTEM DEADLOCK
326	-70	I/O TERMINATED BY CLOSE
327	-71	SPOOL FILES ACTIVE
328	-72	TASK DOES NOT EXIST
329	-73	DEVICE IN USE
330	-74	SYSTEM STACK OVERFLOW
332	-76	DEVICE IN USE
333	-77	ZERO MESSAGE
334	-78	"YOU CAN'T DO THAT"
335	-79	.TOVLD NOT LOADED
336	-80	OPERATOR MESSAGES NOT SYSGEN'ED
337	-81	DISK FORMAT ERROR
338	-82	INVALID BAD BLOCK TABLE
339	-83	INSUFFICIENT SPACE IN BAD BLOCK TABLE
340	-84	CREATE CONTIGUOUS OF ZERO LENGTH
341	-85	PROGRAM IS NOT SWAPPABLE
342	-86	BLANK TAPE

Licensed Material-Property of Data General Corporation

384	-128	ILLEGAL FUNCTION
385	-129	VARIABLE LENGTH BLOCKS NOT ALLOWED
386	-130	REWRITE MODE NOT ALLOWED FOR NON-DISK DEVICE
387	-131	ILLEGAL FUNCTION FOR DEVICE
388	-132	OPEN PROCESSING ERROR
389	-133	FUNCTION DOES NOT PROCESS SPECIFIED RECORD FORMAT
390	-134	FILE ALREADY IN USE
391	-135	FILE CURRENTLY OPENED FOR EXCLUSIVE USE
392	-136	FILE NOT OPEN
393	-137	PERIPHERAL DEVICE CURRENTLY OPENED
394	-138	SPECIAL FILE PROCESSING ERROR
395	-139	UNRESOLVED RESOURCE CONFLICT
396	-140	REWRITE INVERTED WITH CHANGED DATA ADDRESS
397	-141	FILE NAME ALREADY IN USE
398	-142	BLOCK SIZE EXCEEDS WINDOW
399	-143	VIRTUAL MEMORY EXHAUSTED
400	-144	SYSTEM TRANSLATION TABLE LOAD ERROR
401	-145	THE SYSTEM CANNOT OPEN THE FILES AS REQUESTED
402	-146	VOLUME CLOSE ERROR
403	-147	INSUFFICIENT SPACE TO OPEN FILE
404	-148	BEYOND LOGICAL BOUNDS OF FILE
405	-149	TRANSLATION SPECIFICATION ERROR
406	-150	VOLUME DOES NOT EXIST
407	-151	RECORD IS LOCKED AND "HOLD" NOT REQUESTED
408	-152	DISK SPACE EXHAUSTED
409	-153	RECORD BEYOND BOUNDS OF FILE
410	-154	ERROR WHILE CLOSING VOLUME DURING INTERNAL VOLUME SWITCH
411	-155	PHYSICAL I/O ERROR IN ASYNCHRONOUS BUFFER PROCESSING
412	-156	DISK I/O ERROR WHILE FLUSHING BLOCK
413	-157	DISK OR MAGNETIC TAPE TIMEOUT
414	-158	CAN'T OPEN FOR UPDATE WITH UNBLOCKED RECORDS
415	-159	INSUFFICIENT PARAMETERS FOR TRANSLATION
416	-160	SPECIAL FILE OPEN ERROR IN PREOPEN
417	-161	SPECIAL FILE CLOSE ERROR IN PREOPEN
418	-162	INTERNAL SPECIAL FILE CLOSE ERROR
419	-163	RDOS OPEN FAILURE
420	-164	FILE NAME ALREADY IN USE
421	-165	ZERO LENGTH TRANSFER REQUEST ON DISK DEVICE
422	-166	TOO MANY ISAM AUTO-UPDATE FILES
423	-167	INDEX NAMING ERROR
424	-168	INDEX NOT IN DATA BASE INDEX DEFINITION FILE
425	-169	FILE NAME TOO LONG
426	-170	ISAM FILE SPACE EXHAUSTED
427	-171	MAG TAPE PROCESSING ERROR
428	-172	SYSTEM DOES NOT SUPPORT DEVICE
429	-173	END OF VOLUME FOR OUTPUT FILE
430	-174	END OF VOLUME FOR INPUT FILE
431	-175	DUPLICATE KEY ENCOUNTERED IN INTERNAL PROCESSING
432	-176	INTERNAL COMPRESSES KEY PROCESSING ERROR
433	-177	ILLEGAL RELATIVE MOTION
434	-178	INVALID NODE ADDRESS ENCOUNTERED INTERNALLY
435	-179	INVALID ENTRY ADDRESS ENCOUNTERED INTERNALLY
436	-180	TOP LEVEL ERROR
437	-181	SUBINDICES NOT ALLOWED
438	-182	SUBINDEX NOT PRESENT
439	-183	(SUB)INDEX BOUNDARY ENCOUNTERED
440	-184	DELETE POSITIONING ERROR
441	-185	MULTIPLE KEY WRITE ERROR
442	-186	KEY TOO LONG OR OF ZERO LENGTH
443	-187	INVALID ENTRY NUMBER ENCOUNTERED INTERNALLY
444	-188	NEITHER 'KEYED' NOR 'RELATIVE MOTION' SPECIFIED
445	-189	KEY ALREADY EXISTS AND DUPLICATES NOT ALLOWED
446	-190	KEY POSITIONING ERROR
447	-191	ILLEGAL RECORD LENGTH
448	-192	CLI Error Messages (following INFOS Error Messages)
472	-216	
512	-256	NO DATA BASE RECORD FOR INDEX ENTRY

Licensed Material-Property of Data General Corporation

INFOS Error Messages (continued)

513	-257	NODE SIZE TOO BIG FOR PAGE
514	-258	NODE SIZE WILL NOT HOLD 3 ENTRIES
515	-259	DATA BASE RECORD IS LOCKED
516	-260	SUBINDEX IN USE ENCOUNTERED DURING PROCESSING
517	-261	FILE AND SYSTEM VERSIONS INCOMPATIBLE
518	-262	TOO MANY SUBINDEX LINKS
519	-263	SUBINDEX ALREADY PRESENT
520	-264	SUBINDEX LEVEL LIMIT EXCEEDED
521	-265	INDEX ENTRY WITH SUBINDEX MAY NOT BE DELETED
522	-266	PHYSICAL DELETE ACCESS MUST BE 'KEYED'
523	-267	INDEX ENTRY IS LOCKED
524	-268	WRITE ACCESS MUST BE 'KEYED'
525	-269	ILLEGAL LABEL FORMAT ON MAG TAPE
526	-270	ILLEGAL LABEL SPECIFICATION
527	-271	VOLUME IDENTIFIERS DO NOT MATCH
528	-272	FILE IDENTIFIERS DO NOT MATCH (INTERNAL ERROR)
529	-273	FILE SEQUENCE NUMBERS DO NOT MATCH (INTERNAL ERROR)
530	-274	FILE GENERATION NUMBERS DO NOT MATCH (INTERNAL ERROR)
531	-275	FILE EXPIRATION DATE NOT YET REACHED
532	-276	BLOCK COUNT ON TRAILER LABEL DISAGREES WITH ACTUAL COUNT
533	-277	RECORD FORMATS DO NOT MATCH
534	-278	INCORRECT FILE SECTION NUMBER ON HEADER LABEL

CLI Error Messages

448	-192	NOT ENOUGH ARGUMENTS
449	-193	ILLEGAL ATTRIBUTE
450	-194	NO DEBUG ADDRESS
451	-195	COMMAND LINE TOO LONG
452	-196	NO STARTING ADDRESS
453	-197	CHECKSUM ERROR
454	-198	NO SOURCE FILE SPECIFIED
455	-199	NOT A COMMAND
456	-200	ILLEGAL BLOCK TYPE
457	-201	NO FILES MATCH SPECIFIER
458	-202	PHASE ERROR
459	-203	TOO MANY ARGUMENTS
460	-204	TOO MANY ACTIVE DEVICES
461	-205	ILLEGAL NUMERIC ARGUMENT
462	-206	FATAL SYSTEM UTILITY ERROR
463	-207	ILLEGAL ARGUMENT
464	-208	IMPROPER OR MALICIOUS INPUT
465	-209	TOO MANY LEVELS OF INDIRECT FILES
466	-210	SYNTAX ERROR
467	-211	BRACKET ERROR
468	-212	PAREN ERROR
469	-213	UNMATCHED < >
470	-214	ILLEGAL NESTING OF < > AND ()
471	-215	ILLEGAL INDIRECT FILENAME
472	-216	ILLEGAL NESTING OF () AND []

End of Appendix

Licensed Material-Property of Data General Corporation

Appendix D
Calling an Assembly Language Subroutine from Business BASIC (UCALL)

If your system is generated to include user-written assembly language subroutines in USERSUBS, you can call a subroutine written is assembly language from a BASIC program. The format of the BASIC call is:

UCALL sub# *[,A_1, ..., A_n]*

Where:

sub# is a numeric expression evaluating to a positive integer (in the range 0 to 32767) representing the subroutine number.

$A_1, \ldots, A_n$ are optional arguments to be passed to the subroutine (*n* must be in the range 1 to 8) and may be arithmetic variables or expressions, or string variables or expressions. Arrays are not supported as arguments. Therefore, dimensioned numeric variable names should include subscripts. (Statement numbers are not permitted as arguments.)

Character String Storage and Definitions

You must refer to the following information if you wish to handle character strings in a UCALLed subroutine. Business BASIC keeps a count of the number of characters currently defined in each string variable (referred to as the current length of the string variable). The current length is stored as part of the header immediately preceding the contents of each string variable. (See Figure D-1.) You must update the current length each time characters are added to or taken away from the string variable.

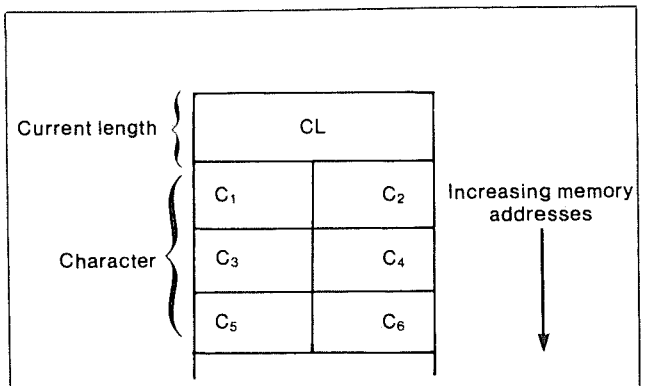


Figure D-1. String Variable Storage

In the following examples, assume that A$ is dimensioned to 10, and A$ = "ABCDE". The current length of A$ is 5.

A *substring* is any contiguous part of a string variable. For example:

A$(2,4) and A$ are substrings of A$

The *current length of a substring* is the number of defined characters within the substring. For example, the current length of A$(4,7) is 2, if only A$(4,4) and A$(5,5) are defined.

The *maximum length of a substring* is the number of character positions within the substring. For example, the maximum length of substring A$(4,7) is 4.

Licensed Material-Property of Data General Corporation

Linking the Assembly Language Subroutine

Improper use of assembly language subroutines, system calls, or task calls can crash the system.

Assembly language subroutines must be submitted to the system manager at system load time. The subroutines in a file named USERSUBS.RB are input to the relocatable loader when the BASIC system save file is created. You must include a subroutine table with your subroutines. The table must have the entry point USERSUBS.

The subroutine table is a list of all assembly language subroutines available to a BASIC program. For each assembly language subroutine, a four-word list is required in the table containing the following:

- subroutine number
- subroutine entry point
- number of arguments
- argument control word

You terminate the table by using a subroutine number of -1.

The argument control word is used by BASIC to give runtime error checking on the types of arguments. The control word is divided into either two-bit fields for the eight possible arguments A_1, ...A_8. The value of the two-bit field determines the allowable argument either passed or returned.

00 argument may be any string expression
01 argument must be a string variable
10 argument may be any numeric expression
11 argument must be a numeric variable

The argument control word is written in an assembly language program such that the arguments are connected by a plus sign (+) and are described as shown in Figure D-2.

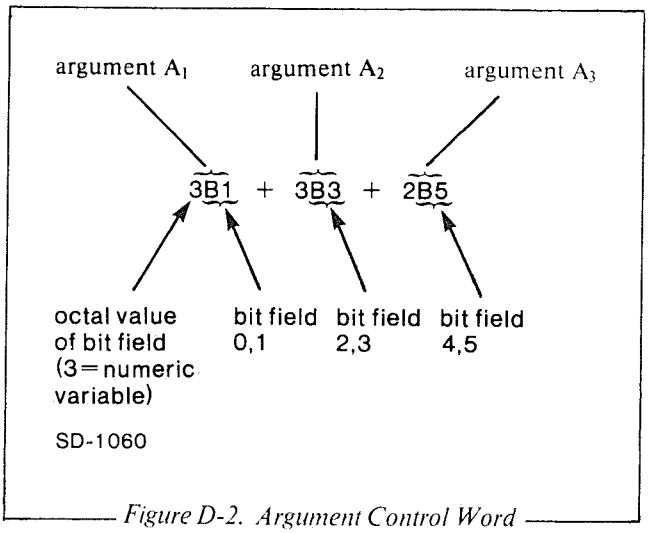


Figure D-2. Argument Control Word

BASIC calls the assembly language subroutines by the sequence:

```
        LDA     2,.+2     ;AC2 POINTS TO TOP
                          ;OF ADDRESS LIST
        JMP     <SUB>     ;JMP TO ASSEMBLY
                          ;LANGUAGE SUBROUTINE
        ADLST

ADLST:  <arg A1>
        <arg A2>
        .
        .
        .
        <arg An>
        JMP     BASIC     ;RETURN TO BASIC
                          ;INTERPRETER
```


Licensed Material-Property of Data General Corporation

If A_n is a substring of a string variable, the address list contains the address of the string descriptor words, which contain the following information:

word 1: byte address of the first character of the substring

word 2: current length of the substring

word 3: maximum length of the substring

word 4: word address of the current length of the string variable

If A_n is a string expression, the address list contains the address of the string descriptor words, which contain the following information:

word 1: byte address of the first character of the string

word 2: length of the sting

If A_n is a numeric variable, the address list contains the storage address of the variable. (All numeric variables are represented in standard floating-point format.)

If A_n is a numeric expression, the address list contains the storage address of the value of the expression.

The following example shows legal and illegal calls to a subroutine, the subroutine table, and the subroutine itself. The argument list in a BASIC call to this subroutine must match the argument control word specified in the subroutine table.

An illegal UCALL will result from an attempt to pass a variable in the UCALL that does not have a previously assigned value. All variables passed in the UCALL must have been assigned values even if their current value will not be used in the UCALLed subroutine.

Several subroutines are available in BASIC to help you manipulate numbers and character strings. The pointers to the routines are in page zero and and should be declared as displacement externals.

```
                        .TITLE  SBRTB    ; BASIC ASSEMBLY LANGUAGE SUBROUTINES
                        .ENT    SBRTB    ; ENTRY POINT : SBRTB
                        .NREL            ; NORMAL RELOCATABLE CODE

; SUBROUTINE TABLE

SBRTB:      1                            ; SUBROUTINE NUMBER
            SUB1                         ; SUBROUTINE ENTRY POINT
            2                            ; NUMBER OF ARGUMENTS
            3B1 + 3B3                    ; ARGUMENT CONTROL WORD, BOTH ARGS ARE
                                         ; NUMERIC VARIABLES
            -1                           ; END OF TABLE

SUB1:
; CALLING SEQUENCE:  CALL 1,A,B
; THIS ROUTINE IS THE EQUIVALENT OF LET B = A
; THIS ROUTINE IS NOT REENTRANT
            STA         2,RET            ; SAVE ADDRESS LIST
            LDA         3,0,2            ; ADDRESS OF ARG 1
            LDA         3,0,3            ; WORD 1 OF ARG 1
            LDA         2,1,2            ; ADDRESS OF ARG 2
            STA         3,0,2            ; WORD 1 OF ARG 1 TO WORD 1 OF ARG 2
            LDA         2,RET            ; ADDRESS LIST
            LDA         3,0,2            ; ADDRESS OF ARG 1
            LDA         3,1,3            ; WORD 2 OF ARG 1
            LDA         2,1,2            ; ADDRESS OF ARG 2
            STA         3,1,2            ; WORD 2 OF ARG 1 TO WORD 2 OF ARG 2
            LDA         3,RET            ; ADDRESS LIST = RETURN ADDRESS -2
            JMP         2,3              ; RETURN TO BASIC (2 = NO. OF ARGS)
RET:        .BLK        1
            .END
```

Figure D-3. Example of an Assembly Language Subroutine

End of Appendix

Licensed Material-Property of Data General Corporation

Index

Within this index, *f* or *ff* after a page number means *and the following page* (or *pages)*. In addition, primary page references for each topic are listed first.

(*) prompt 1-1ff, 1-7
000: prompt 7-11ff
  (see also EDIT utility)
(?) prompt for INPUT 2-5, 3-34f
& (external routines)
@ (files used)
$ (external storage)
! (module name) 8-36
> (calling sequence)
(description)
% (External Programs)
  PRTCOM 8-29
@ (-30) - form feed 3-66
@ (num) - cursor positioning and terminal control 3-34f, 3-36f, 3-65ff, 3-68f, 3-87ff
(!) prompt for BASIC CLI 1-7
> (right-angle prompt for OPCLI) 10-15
(-) and (*) conventions 3-93f

.A (keyboard edit command) 1-4f
A, EDIT utility command 7-12, 7-18
  (see also file input/output, EDIT utility)
A (print format) 3-70
Aw 5-9
  (see also field format code, crammed string, crammed field)
AA account 10-10f (system manager account)
  (see also System Security)
AB account 10-11 (system programmer)
  (see also System Security)
ABS 3-4
absolute value 2-3, 3-4
ACC, EDIT command 7-13, 7-18
  (see also control characters)
access, direct
  (see Random File Access)
access, file 3-60f, 3-73f, 4-4, 4-19ff, 4-26, 4-39
  (see also subfile, file characteristics)
access, to directories 10-13
access, to file, keywords 4-26ff
access, file statements 2-3
access, linked-available-record 4-30ff, 5-1
access modes, file 3-60f, 4-16, 7-35
access, random 3-31, 3-60f, 3-64f, 3-73f, 3-104f, 4-16, 4-28ff, 5-1, 5-5
access, sequential 3-31, 3-34, 3-60f, 3-73f, 3-104f 4-16, 4-26ff,
account 1-1, 3-51, 3-90, 7-36, 10-10ff
account code 7-36
  (see also PORTS utility)
account ID 7-41, 10-11
account name 1-1, 3-51, 3-90
account number 5-1
ACCOUNTING file 10-10ff, 10-13
  (see also System Security)
accounts logged on 7-36
  (see also PORTS utility)
ACL, EDIT command 7-13, 7-18
  (see also lowercase characters)
add a new record 5-4
add comments to programs 7-11ff
  (see also EDIT utility)
ADD key (FM template) 5-3f, 5-10
  (see also File Maintenance)
add key to index file 3-38f, 4-35
addition 2-5
alignment pattern file 8-42
ALIGNVFU file 8-42
ALL (OPCLI command) 10-15
allocate new record 4-32
  (see also GETREC.SL)
allocation, dynamic 3-74, 3-104f, 4-25, 5-1, 5-5, 7-25 7-31, 7-40, 7-47
ALM multiplexor line 7-33, 10-4
  (see also LSPEED utility, baud rate)
alphanumeric fields, edited 5-8
alphanumeric fields, unedited 5-8
alphanumeric key, unsigned 7-39
alternate input file 7-12
  (see also EDIT utility)
AND, logical 3-5, 3-53ff, 3-62f, 3-84
angle brackets 2-6, 3-34f, 3-36f, 6-2
append a string to a string 2-6
append a file 8-2
APPEND command 8-2
approximate match (key) 3-41f, 4-7, 4-35, 9-1
  (see also Index File)
argument 2-1, Preface
arithmetic 2-5
arithmetic functions 2-2, 2-8
arithmetic operators 2-5
array 2-4f, 2-1, 3-20f, 3-44, 3-73f, 3-104f
array, column of 2-4f, 3-20f
array element 2-4f, 3-20f, 3-44, 3-104f

Licensed Material-Property of Data General Corporation

array, redimension 2-5, 3-20f
array, rows and columns 2-4f, 3-20f
array variables 2-4f, 2-3, 3-20f, 3-44, 3-73f, 3-104f
array variables, dimensioning 3-20f
arrays, one dimensional 2-4f, 3-20f, 3-44
arrays, two dimensional 2-4f, 3-20f, 3-44
arrays, quick definition of 5-14
(see also TABBUILD utility)
ASC function 3-7, 2-6, 3-12, 4-17
ascending order (sort) 7-21, 7-39
ASCII character code 2-6, 3-7, 3-12, 3-25, 3-67f 3-86ff, 3-93, 7-43, 7-46, 8-45, Appendix A
ASCII characters 1-2, 2-5ff, 3-7, 3-12, 3-25, 3-65ff 3-68f, 3-86ff, 3-93, Appendix A
ASCII/EBCDIC translations 3-93
ASCII listing (source file) 3-11, 3-22, 3-75
ASG command 8-3
assign a device 8-3
assign binary number to string 2-6, 3-7, 3-12, 3-44
assign digits to string 2-6, 3-7, 3-12, 3-44
assignments, string 2-5ff, 3-44
assignments, substring 2-6f, 3-44
assign reserved file 3-94f
assign values to variables 2-3, 3-17, 3-44
asterisk and dash conventions 3-93f, 7-13, 8-1
asterisk prompt 1-1ff, 1-7, 2-1
asynchronous (ALM) driver support 10-1
(see also ALM multiplexor line)
asynchronous multiplexor (ALM) 10-1
ATTACH utility 7-2
(see also Subroutines and Utilities)
attached job 3-94
attach terminal to de-attached job 7-2
attributes 4-3ff, 8-6ff
(see also File Attributes)
attribute protected 4-3, 8-6
audible alarm, terminal 3-68
AUTO key 5-4, 5-11
(see also File Maintenance)
automatic display 5-4, 5-11
automatic job execution 10-14
automatic text justification 7-6, 7-8
(see also DOC utility)
automatic lock flag 3-38ff, 3-41f, 7-32
automatic locking of index files 4-6f
(see also File Access)
automatic log-off 10-11

B, EDIT command 7-14, 7-18
Bw.d 5-9
(see also field format code, bit flag field)
background 10-3f
backslash key 1-3, 3-89, 7-6ff
Backtab key
(see also F12 function key, File Maintenance)
BASIC CLI 2-2f, 7-3, 8-1
exiting 8-30
BASIC CLI commands (Chapter 8), 8-1ff
(see also CLI commands)
BASIC CLI command, executing 1-7, 7-3
BASIC.ER file 3-24, 3-56, 3-98, 7-45
BASIC.PS file 10-10
(see also generating push space)
BASIC.SW file (swapping) 10-10
baud rate (LSPEED) 7-33
(see also LSPEED utility, ALM multiplexor line)
beginning of tape marker 3-54
bell, terminal 3-68
binary data, input from file 3-73f
binary data, output to file 3-104f
binary field (B) 5-8
(see also field formats)
binary number 2-3
binary representation, triple, double and single precision numbers 2-3f
binary string 4-3ff, 7-18
(see also records)
binary value in string 2-6, 3-7, 3-12
bit flags 3-52ff, 7-44
bit flag field (F) 5-8
(see also field formats)
bit positions 3-6, 3-53, 3-62, 3-84
bits 2-3ff, 2-6, 3-5, 3-52ff, 3-62, 3-84
bit shift, logical 3-84
blank lines 7-6
blank lines, printing 3-66
BLDCOM command 8-3
documentation file 8-29
blinking field 3-68
block input/output 3-8, 3-10, 4-33f
block number, starting 3-8, 3-10
block mode, set 7-44
BLOCK READ 3-8, 3-10, 4-28
BLOCK READ FILE 3-8, 2-3, 3-10, 4-33f
block transfer 3-8, 3-10
BLOCK WRITE 3-10, 4-28
BLOCK WRITE FILE 3-10, 2-3, 4-33f
block zero of index file 3-39, 4-6f, 4-18ff, 7-31f
blocking factor 3-39, 4-7, 4-19ff, 4-25, 7-26, 7-28f 7-31f, 7-47
blocks (sectors) 4-6f, 4-19ff, 4-28ff, 8-11
(see also Index Files)
blocks at level 0, number 7-29
blocks, contiguous disk 4-4, 8-5, 8-8
(see also Contiguous Files)
blocks, input from file or common area 3-8
BR, EDIT command 7-18
brief mode, EDIT utility 7-18
BSG and generation 10-1 to 10-9
BSG program, RDOS/DOS 10-2
BSG sample dialog 10-3 to 10-9
buffer, edit 1-4, 3-45f, 7-11ff, 7-17
buffer, PRINT USING 3-70f
buffer string, index 3-38ff, 3-41f, 4-35, 4-38f
buffer string, INFOS packets 9-1ff
buffers 10-1
build a filename or command file 8-4
(see also BUILD)

Licensed Material-Property of Data General Corporation

build C1 array, find subfile 7-22
(see also FINDFILE.SL)
BUILD command 8-4
build index file 7-28, 7-47
(see also INDEXBLD utility, XBUILD utility)
build index file, from sorted tag or data file 7-26f
(see also IBUILD)
build modules diskette 10-1
build record 0 7-31
(see also INITFILE utility)
Business BASIC, executing 10-10
BYE 3-11, 1-7
BYE (OPCLI command) 10-15
byte offset into record 7-37
(see also POSFL.SL subroutine)
byte offset into file 3-38ff, 3-41f, 3-64f, 3-73f
byte of lock area, starting 3-47f
byte offset, to record 0 4-11ff, 7-31f, 7-40
byte position (POSFL.SL) 4-4, 7-37
(see also file characteristics, record position)
byte position (pointer), file 3-8, 3-10, 3-31, 3-61
3-64f, 3-73f 3-104f, 4-29ff, 4-30ff
byte position, string and substring 2-6f, 3-12, 4-17,
4-19
byte relative to byte 0 3-64f, 3-73f
bytes 2-3ff, 2-6f, 3-7, 3-12, 3-16, 3-64f, 3-73f
3-104f
byte transfer 3-7, 3-73f, 3-83, 3-104f, 4-6
byte zero of file 3-64f

.C (keyboard edit command) 1-4
CCONT command 8-5
CDIR command 8-5
C, EDIT command 7-15, 7-18
(see also changing strings with EDIT utility)
C1 (file characteristics) array 3-39f, 3-41f, 4-4
4-9ff, 4-12ff, 4-17ff, 4-30ff, 4-35ff, 7-4, 7-22, 7-25
7-35, 7-37
C1 array, build, find subfile 7-22
(see also FINDFILE.SL)
calculate exteded percentage 7-5
(see also DIV.SL)
calculate percent of a number 7-34
(see also MUL.SL subroutine)
calculate index information 7-29
(see also INDEXCALC utility)
calculations, perform 2-3, 3-44, 3-65
calling assembly language subroutines Appendix D
carriage return Preface, 1-1, 1-3f, 5-4, 5-10
carriage return terminator 3-36f, 3-66ff, 3-70f
cassette input/output 3-52ff
card reader ($CDR) 4-1f
CCONT command 4-9ff, 5-2
$CDR (card reader) 4-1f
cells 10-1
centering lines 7-6f
(see also DOC utility)
CHAIN 3-11f, 1-7, 2-1f, 3-81, 3-97
CHAIN, CLI command 8-6
chain of deleted records 4-32
(see also DELREC.SL)
change a record 5-3f
CHANGE key (FM Template) 5-3f, 5-10f
(see also File Maintenance)
CHANGE PAGE key 5-3f, 5-7f, 5-11
changing file attributes 8-6, 10-11
change filename 3-76
change link access attributes 8-7
change terminal key functions 7-43
(see also TERM utility)
changing strings, with EDIT utility 7-15
(see also C, EDIT command)
channel assignments, display 7-43
(see also UCHANS utility)
channel, find available 4-15ff, 7-35
(see also OPEN utility)
channel, next available 7-22
channel number of a file 3-60f, 3-8, 3-10, 3-14,
3-22, 3-34, 3-36, 3-38ff, 3-41f, 3-52, 3-64f, 3-70
3-73f, 3-98, 3-104f, 4-4, 4-12f, 7-32, 7-35, 7-37,
7-41
channels 10-1, 10-5
chapters document produced by DOC utility 7-6
character code, ASCII 2-6, 3-7, 3-12, 3-67f, 3-86ff,
3-93, Appendix A
character delete echo 3-67, 3-87, 7-43
character delete key 3-67, 3-87, 7-43
character format file 2-8, 3-22, 3-34, 3-45, 3-65ff,
3-68f, 7-11, 7-17f, 8-29, 8-41
(see also text, listing and source files, EDIT utility)
character format program source statements 3-22,
3-45
characters, ASCII 1-2, 2-5ff, 3-7, 3-12, 3-45, 3-65ff,
3-68f, 3-70f, 3-86ff, 3-93, Appendix A
characters, control 3-89, 7-13
(see also EDIT utility, ACC, EDIT command, DCC
EDIT command)
characters, for formatted input/output 6-4ff
characters, lowercase 3-89, 3-93, 7-13
(see also EDIT utility, ACL, EDIT command, DLC,
EDIT command)
characters per line maximum for EDIT disk
buffer 7-11
(see also MAXn EDIT command)
CHATR command, BASIC CLI 4-3, 8-6, 10-11
CHLAT command, BASIC CLI 4-3, 8-7
CHR$ 3-12f, 2-6, 3-7
clean up linked data file 7-40
(see also RELINK utility)
clear an error 5-4
CLEAR ERROR key (FM Template) 5-3, 5-10f
(see also File Maintenance)
clear working storage 3-56
CLI, BASIC commands 1-7, 2-2, 7-3
CLI, BASIC utility 7-3
CLI.CM 10-2

Licensed Material-Property of Data General Corporation

CLI, COM file, read 7-24
(see also GETCM.SL subroutine)
CLI commands, BASIC (Chapter 8), 8-1ff
APPEND 8-2
ASG 8-3
BLDCOM 8-3
BUILD 8-4
CCONT 8-5
CDIR 8-5
CHAIN 8-6
CHATR 8-6
CHLAT 8-7
CPART 8-8
CRAND 8-9
CREATE 8-9
DELETE 8-10
DIR 8-11
DISK 8-11
DUMP 8-12
EQUIV 8-13
FDUMP 8-13
FILCOM 8-14
FILES 8-15
FLOAD 8-15
FPRINT 8-16
FREE 8-17
GDIR 8-17
GSDIR 8-18
GSYS 8-18
GTOD 8-19
INIT 8-19
LIBRARY 8-20
LINK 8-21
LIST 8-22
LOAD 8-24
LSTCOM 8-25
LSTMERGE 8-26
MDIR 8-27
MOVE 8-27
POP 8-28
PRINT 8-29
PRTCOM 8-29
QUIT 8-30
RELEASE 8-31
RENAME 8-31
SDIR 8-32
SLINE 8-32
SPDIS 8-33
SPEBL 8-33
SPKILL 8-34
STAT 8-35
TABLE 8-37
TCOPY 8-38
TFER 8-39
TPRINT 8-40
TYPE 8-41
UNLINK 8-41
VFU 8-42ff
XFER 8-45
CLI commands, generating 7-24, 8-1
(see also GETCM.SL subroutine)
CLI execution, immediate (!) 7-3
CLI, RDOS/DOS 2-2, 7-24
CLI, RDOS/DOS commands 2-2
CLI, swaps to 7-3
CLOSE and CLOSE FILE 3-14
CLOSE, NFOS 9-2
closing opened files 3-14
colon (:) comments Preface, 2-8, 7-17f, 8-3
column, array 2-4f, 3-20f
column counter 3-66ff, 3-71, 3-89
column pointer 3-71
columns, print zone 3-64, 3-66f, 3-68f
COM, EDIT command 7-17f, 7-19
COM file, CLI, read 7-24
(see also GETCM.SL subroutine)
command 2-1, 1-2
command line 6-6
Command Line Interpreter (CLI) 2-2f, 2-1
command line, RLDR 10-2
command mode, for EDIT utility 7-11ff
commands, BASIC CLI (Chapter 8), 8-1ff, 1-7, 2-2
commands, DOC utility 7-6ff
commands, RDOS/DOS CLI 2-2
commands, keyboard editing 1-4, 3-45
commas in INPUT 3-34f
commas in PRINT 3-66ff
comment mode 7-11, 7-17f
(see also EDIT utility, COM, EDIT command)
comments, add to programs 7-11ff
(see also EDIT utility)
comments, colon in program listings Preface, 2-8, 7-17f, 8-3, 8-29
common area 1-7, 3-8, 3-10, 3-12, 4-15, 7-3, 8-1, 8-28, 8-30
common area 4-2,4-33F, 7-24, 8-1
compact output 3-66
comparing files 8-14
comparing numeric expressions 3-49, 3-62f
comparing strings 3-32, 3-93f
complete function 5-4
compile format 7-11, 8-6
(see also save file)
compute bound time slice 10-15
compute priority 10-11, 10-16
CON 3-15, 1-6f
concatenated keys 3-39, 4-7, 4-25
concatenated strings 4-35
concatenation operator (comma) 2-6
conditional paging 7-6ff
(see also DOC utility, \CPAGE command)
conditional transfer 3-32f, 3-57f
constants 2-1
contiguous files 4-3, 4-4, 4-9ff, 4-18ff, 5-1, 8-5
(see also File Attributes)
contiguous file organization (C) 4-3ff
continue execution of a program 3-15, 3-81

Licensed Material-Property of Data General Corporation

contiguous disk blocks 4-4, 8-5, 8-8
(see also contiguous files)
control characters 3-89, 7-13
(see also EDIT utility, ACC, EDIT command, DCC, EDIT command)
control of magnetic-tape/cassette 3-52ff
control record 5-4ff
(see also File Maintenance)
control variable 3-26f
conventions, dash and asterisk 3-93f
conventions, documentation Preface
conventions, for EDIT 7-13, 8-1
conversion, (of) unsigned integer 7-46
convert string to a number 7-46
(see also VAL.SL)
copy buffer (PRINT USING) 3-70
COPY, EDIT command 7-12, 7-19
copying lines with EDIT utility 7-15f
(see also DITTO, EDIT command)
core image file
(see save or program file)
correct sign of numeric after I/O 3-63, 3-74, 3-83, 4-29ff
\CPAGE, DOC command 7-6ff
(see also DOC utility, conditional paging)
CPU switch settings 3-98
CR (carriage return)
(see carriage return)
crammed field (C) 5-8, 6-4
(see also field formats)
crammed keys 3-16, 3-38ff, 3-41f, 3-103, 7-30
crammed string 3-16, 3-90, 3-103, 4-35
cram 3 bytes of string into 2 bytes 3-16, 3-103
CRAND 8-9, 4-10, 5-1f
CREATE 8-9
create a sequential file 8-9
create files 4-9ff
create a contiguous file 8-5
create a random file 8-9
create text files 7-11ff
(see also EDIT utility)
CRM$ 3-16, 3-90, 3-103, 4-35
CSLICE (OPCLI command) 10-15
CTRL-D 7-2
CTRL-X key 1-3
current line for EDIT utility 7-11ff, 7-13f, 7-17
(see also prompt, 000:)
current location of program, indicators 7-36
current program indicators 7-36
cursor positioning 3-34f, 3-36f, 3-65ff, 3-68f, 6-2, 6-5
cursor positioning keys 6-2, 6-5

D, EDIT command 7-16, 7-19
(see also deleting lines with EDIT utility)
Dw.d (print format) 3-70
dash and asterisk conventions 3-93f
DATA 3-17, 2-3, 3-72f, 3-79
Data Base Access Method (DBAM) Chapter 9
database inversion 9-1
data file (save file without extension) 4-2, 7-28, 7-42
data file printing, FM 5-14
(see also FMPRINT utility)
data file, initialize 7-31
(see also INITFILE utility)
data file record number 3-38ff, 3-41f, 3-73f
data file records 5-6
data file password 5-7
data files 2-8, 3-73f, 4-6f, 4-18ff, 4-20ff, 4-34, 5-1, 5-7
data input/output, magnetic-tape/cassette 3-52ff
data length 3-98
data list for DATA and READ 3-17, 3-72f, 3-79
data list pointer 3-79
data records 4-5, 7-42
data transfer, variables 2-3f, 3-73f, 3-104f, 4-26ff, 4-29ff
data types for input/output 6-4ff
date, julian 3-91f
dates 3-91f, 3-98, 7-7f, 8-19, 10-14
day of month 3-98
DBAM (Data Base Access Method) Chapter 9
DCC, EDIT command 7-13, 7-19
(see also control characters)
deadly embrace 3-48, 4-39
de-attach key 3-87, 7-2, 7-43
de-attached job 3-94, 3-98, 7-1, 7-36, 10-11
decimal ASCII character code 2-6, 3-25, Appendix A
decimal, floating 6-4
decimal number 2-3
decimal points 2-3, 3-70f, 6-4, 7-46
decimals, varying number of 7-44
decisions (conditional transfers) 3-32f
DEF 3-18
default characters for crammed strings 3-16, 3-90, 3-103
default columns per line 3-64, 3-100
default for arrays 3-20f
default format 5-6
default line speed 10-4
default margins for text input files 7-6f
(see also DOC utility)
default output device 3-90
default overlays 10-6
default page 5-6
default parity, tape 3-52ff
default tab setting 3-66, 3-100
default terminal characteristics, reset 3-67, 3-86
define a program loop 3-26f
define/delete a subindex (INFOS) 9-2f
define a screen 6-1ff
(see also Screen Maintenance (SM))
define your own function 3-18
DELAY 3-19
delay execution 3-19
DELETE command 3-19, 8-10
delete a file from a directory 3-19, 8-10
delete a key from an index 3-40ff, 4-36
DELETE CHAR (SHIFT-F2), SM function key 6-2f, 6-6

Licensed Material-Property of Data General Corporation

DELETE LINE (SHIFT-F4), SM function key 6-2f, 6-6
DELETE, NFOS 9-2
delete record 4-4, 4-12ff, 4-30ff, 4-32, 5-3f, 7-3
(see also DELREC.SL)
DELETE terminal key 1-3, 3-67, 3-87, 5-3f, 5-11
deleted record chain 4-5, 7-4, 7-25, 7-40
(see also Record 0)
deleted records, test status 7-21
(see also FILESORT utility)
deleting lines with EDIT utility 7-16
(see also D, EDIT command)
delimiter, of string 7-13, 7-15
DELREC.SL 3-29, 4-4ff, 4-12ff, 4-30ff, 7-4, 7-25
DELSI, NFOS 9-3
descending order (sort) 7-21
descriptor string, index 3-38ff, 3-41f, 4-34f, 7-32
device, default output 3-90
device-independence 8-13
device, free a 8-17
device or disk file, open for access 3-60f
device, rename 8-15, 8-45
device, reserved 3-94f, 8-3
devices, filenames 4-1f, 8-19
DFSI, NFOS 9-2
digit subfield (N) 5-8
(see also field formats)
digits in string 2-6
DIM 3-20f, 2-4f, 2-6f, 3-25
dimension, array 2-4f, 3-20f
dimensioning arrays and strings 3-20f
DIR command 8-11, 10-13
directories, restricting access 10-13
directory 4-2, 8-11, 8-15, 8-17ff, 8-27
directory, changing 8-11
(see DIR)
directory, current name 3-90, 8-17, 8-18
directory, deleting 8-10
directory, delete a file 3-19, 8-10
directory format 10-2
directory, library 4-10, 8-22
directory, link entries 4-2
directory, master 8-27
directory, rename file entry 3-76
directory specifier 10-11
directory, system 8-18, 8-32, 10-10f
disable column counter 3-89
disable interrupts (IKEY) 3-59
DISK command 8-11
disk, contiguous blocks 4-4
(see also contiguous files)
disk file 3-60f, 8-39
disk files, copy to output file 8-12
copy one file to another 8-45
disk priority 10-11
disk space management 4-4
(see also dynamic record allocation)
diskettes, loading from 10-1
disks, partitioning 4-2, 4-3
disks, partitions 4-2, 8-11
disks, subpartitions 4-2
disks, user-directories 4-2
Diskette Operating System 1-1
disk file, sort 7-21
(see also FILESORT utility)
disk space 8-11
displacement 8-14
DISPLAY SM command 6-6
display channel assignments 7-43
(see also UCHANS utility)
display current directory name 8-17
display current locks 7-33
(see also LOCKS utility)
display current system name 8-18
display fields 5-6
display files 8-41, 8-43
display jobs on system 7-36
(see also PORTS utility)
Display key 5-4, 5-11
(see also File Maintenance)
display names of file 8-15
in library directory 8-20
display page 5-6
display record 5-4
display system channel assignments 7-41
(see also SCHANS utility)
display time and date 8-19
DITTO, EDIT command 7-16, 7-19
(see also copying lines with EDIT utility)
DIV.SL subroutine 7-5
(see also Subroutines and Utilities)
dividend, calculation of PCNT(X) 7-5
(see also DIV.SL)
division 2-5
divisor, calculation of PCNT(Y) 7-5
(see also DIV.SL)
DLC, EDIT command 7-13, 7-19
(see also lowercase characters)
DOC.HD file 7-6
(see also DOC utility)
DOC utility 7-6ff
DOC utility commands 7-6ff
\CPAGE 7-7
\ECPAGE 7-8
\EQC 7-8
\ETX 7-8
\HEAD 7-7, 7-8
\MAR 7-7, 7-8
\NLF 7-7, 7-8
\PAGE 7-7
\SPAGE 7-8
\STX 7-8
\QC 7-8
DOC utility, example 7-6ff, 7-9f, 7-11
DOCTOC utility 7-11
(see also DOC utility, print table of contents)
documentation conventions Preface
documentation source file (.DS) 4-2
dollar sign in variable name 2-3, 2-6, 7-40

Licensed Material-Property of Data General Corporation

DOS 1-1, 10-1ff
DOS installation 10-2
double integer (binary) filed (D) 5-8
(see also field formats)
double (0) precision 10-3
double precision binary representation 2-3f
double precision numeric variable 2-3ff, 3-73f, 3-104f
.DR extension 8-5, 8-8, 8-32
.DS file extension 4-1f, 5-12
driver support, asynchronous (ALM) 10-1
dual processor system
(see slave processor)
dummy channel 3-8
DUMP command 8-12f, 8-24
dump format, RDOS 10-1
dump of files
to output file 8-12
fast dump to mag tape 8-13
to line printer 8-16
DUP TO EOL (F1), SM function key 6-2f
duplicate keys 3-39f, 3-41f, 4-6f, 4-19ff, 4-35f, 7-31f, 9-1
dynamic allocation 3-74, 3-104f, 4-4f, 4-25, 4-30ff, 5-1, 5-5 7-25, 7-31, 7-40, 7-47
(see also GETREC.SL)
dynamically allocated file, Record 0 4-30ff, 7-4

.E (keyboard edit command) 1-4f
E (print format) 3-70f
EBCDIC/ASCII translations 3-93, 8-39
echo 3-89
ECLIPSE computer 1-1, 10-1
\ECPAGE, DOC command 7-7f
(see also DOC utility conditional paging)
edit buffer 1-4, 3-45f, 7-11ff, 7-17
edit code 5-8
edit text files 7-11ff, 8-43
(see also EDIT utility)
EDIT utility Preface, 2-8, 3-45, 3-75, 4-28, 7-6, 7-11ff
EDIT utility, command summary 7-18ff
EDIT utility, commands 7-12 to 7-20
EDIT utility, file Input/output 7-12
EDIT utility, specifying input/output files 7-12
EDIT, stopping 7-18
(see also STOP, EDIT command, END, EDIT command, OMIT, EDIT command)
edited alphanumeric fields 5-8
editing a program 1-3ff, 3-45
editing commands, keyboard 1-4, 3-45
element, array 2-4f, 3-20f
enable interrupts 3-59, 3-88ff, 10-10
enable interrupts (IKEY) 3-59
END 3-21, 1-6
END, EDIT command 7-18f
end of file check 3-23, 3-52ff, 3-64f, 3-74
end of file mark for tape 3-52ff
END PROTECT (F5), SM function key 6-2f, 6-6
ENTER 3-22, 1-3, 2-1f, 3-28, 3-45, 7-1
ENTER CMND (F-4), SM function key 6-2f, 6-6
entry point, subroutine 2-2, 3-28f, 3-75, 4-30ff, 7-1
EOF 3-23, 3-64f, 3-74
\EQC DOC command 7-6ff
(see also DOC utility)
EQUIV command 8-13
ERASE 3-23, 1-4
erase program statements 3-23
ERM$ 3-24, 3-56f, 3-98, Appendix C
error 1-6, 3-24, 3-36, 3-56f, 3-98, Appendix C
error codes Appendix C, 3-24, 3-52ff, 3-56f, 3-86 3-98, 7-38, 7-44
error message (.ER) file 4-2, Appendix C
error messages 3-24, 3-56f, 7-45, Appendix C
error messages, SM 6-8, 7-45, Appendix C
error return, INFOS 9-1
error termination code 7-46
error trap 3-24, 3-36, 3-56f
ESCape key 1-1, 1-3, 1-7
\ETX DOC command 7-6ff
(see also DOC utility)
examine reserved file 3-94f
exclamation point 1-7, 7-3
exclusive file access 3-60f, 4-16, 4-39
execute a program 3-81, 3-96, 8-6
execute, utility or another program 3-11, 3-81, 3-96
execute the BASIC CLI 7-3
executing a BASIC CLI command 1-7
executing Business BASIC 10-10
execution, continue 3-15, 3-81
execution, stop 3-96
exponentiation 2-5
expressions 2-1, 3-65
expressions, numeric 2-5, 2-2, 3-49, 3-62f
expressions, relational 2-5
expressions, relational with strings 2-7
extended percentage, calculate 7-5
(see also DIV.SL)
extension, .DR 8-5, 8-8, 8-32
extension, .DS 5-12, 8-5
extension, .IB 10-14
extension, .LS 7-17
extension, .SL 2-2
extension, .Sn 6-2
extension, .SR 7-17
extension, .VF 8-42
extract number from string 2-6, 3-7, 3-12, 3-83, 4-29ff

F, EDIT utility command 7-12, 7-19
(see also specifying input/output files for EDIT)
Fw.d (print format) 3-70f, 5-9
(see also field format code)
F1 to F12 functions keys 5-4
(see also File Maintenance)
FALL (OPCLI command) 10-15
fast sort of file 7-39
(see also QFILESORT utility)

FDUMP command 8-13
field, binary (B) 5-8
(see also field formats)
field, bit flag (F) 5-8
(see also field formats)
field, crammed (C) 5-8, 6-4
(see also field formats)
field definitions, SM 6-14
(see also screen file layout)
field descriptor records 5-5, 5-12ff
field, double integer (binary) (D) 5-8
(see also field formats)
field format code 5-9
field formats 5-8
field, integer (binary), single precision (I) 5-8
field, left-justified 7-44
field, numeric 7-44
field, position cursor to 7-45
field prompt 5-8
field, string, (S) 5-8, 7-44
(see also field formats)
fields, displaying 5-6
fields, edited alphanumeric 5-8
fields, formatted 6-7
fields, high intensity 3-68, 6-1
(see also unprotected fields)
fields, key 4-22f, 4-25, 5-6, 7-21
fields, low intensity 3-68, 6-1
(see also protected fields)
fields, max 5-6
fields, on a CRT screen 3-68, 5-6
fields, protected 6-1, 7-38, 7-44
fields, right-justified 5-8
fields, unformatted 6-7
fields, unprotected 6-1, 7-38, 7-44
FILCOM command 8-14
file access 3-60f, 3-65ff, 3-68f, 3-73f, 3-104f, 4-4, 4-26, 4-39
(see also subfiles, file characteristics)
file access keywords 4-26ff
(see also BASIC statements)
file access statements 2-3
file attributes 4-3ff, 8-6
file attributes, CHATR command 4-3, 8-6
(see also commands, LIST)
file attributes, LIST command 4-3
(see also commands, LIST)
file, BASIC.ER 7-45
file channel number 3-60f, 3-8, 3-10, 3-14, 3-22, 3-31, 3-34, 3-36, 3-38ff, 3-52, 3-64f, 3-70, 3-73f, 3-98, 3-104f, 4-4
file characteristics (C1) array 3-39f, 3-41f, 3-65, 4-4, 4-9ff, 4-12ff, 4-17ff, 4-30ff, 4-35ff, 7-4
file, deleting 8-10
file extensions 4-1f
file extensions, table 4-2
file, with nulls 7-31
(see also INITFILE utility)
file, INFOS Chapter 9
file input/output 2-8, 3-34f, 3-60f, 3-64f, 3-73f, 3-104f
file input/output, EDIT utility 7-12
(see also A EDIT command)
file input/output keywords 2-10f
file lock/unlock device 3-47f
file locking 4-39
File Maintenance (FM) 2-2, 4-6, 4-10ff, 4-18ff, Chapter 5, 5-2
File Maintenance (FM) data file printing 5-14
(see also FMPRINT utility)
File Maintenance function keys 5-4ff, 5-10f
File Maintenance (FM) logging 5-7
File Maintenance (FM) templates 5-3
File Maintenance (FM), using 5-9ff
roll mode 5-9, 5-11ff
ADD 5-3
CHANGE 5-3
CLEAR ERROR 5-3
DELETE 5-3
FIND 5-3
FIND NEXT 5-3
FORMAT CHANGE 5-3
PAGE CHANGE 5-3
STOP 5-3
filename 3-60f, 3-76, 3-81, 4-1ff, 4-15ff, 4-39, 5-5
file, open in access mode 3-60f
file organization 4-3ff, 8-45
contiguous (C) 4-3, 4-4
sequential 4-3, 8-9
random (R) 4-3, 8-9
file, PARAM 4-4, 4-9, 4-11, 5-2, 7-22, 7-28ff, 7-40
file pointer, position to record 7-37
(see also POSFL.SL subroutine)
File Setup 4-9ff
file table, reserved 10-6
file, U$ERS 10-13, 10-11
FILES command 8-15
files 2-1, 2-8, 3-60f, 3-73f, 3-104f, Chapter 4, 8-15
file size 4-12ff, 8-22
(see also record number)
FILESORT utility 7-21
(see also sort disk file)
file status 7-40
file structure, ISAM 1-1
files, byte position 3-31, 3-61, 3-64f, 3-73f, 3-104f
files, character format 2-8, 3-22, 3-34, 3-60f, 7-17
files, closing 3-14, 3-60f
files, common area (.CM) 4-2
files, contiguous 4-3, 4-4, 4-9ff, 4-18ff, 5-1
files, copying between tape and disk 8-39
files, create 4-9ff, 8-22
files, data 2-8, 3-38ff, 3-41f, 3-73f, 3-104f, 4-6f, 4-20ff, 5-1, 5-7, 7-28, 7-42
files, delete from a directory 3-19
files, documentation source (.DS) 4-2
files, dynamically allocated 4-30ff, 7-4
files, error message (.ER) 4-2

Licensed Material-Property of Data General Corporation

files, index 2-8, 3-38ff, 3-41f, 30-61, 3-74, 3-104f, 4-6f, 4-9ff, 4-20ff, 5-1, 5-5, 7-42
files, listing Preface, 2-8, 3-11, 3-60, 4-2, 7-1, 7-11, 7-17f
(see also source and ASCII listing files)
files, loading 8-15, 8-43
files, log 5-1, 5-7, 5-14
files, logical 4-4, 4-9f, 4-12f, 7-4
(see also physical files, subfiles)
files, overlay (.OL) 4-2
files, physical 4-4, 4-13, 7-41
(see also logical files, subfiles)
files, physical, open 7-35
(see also OPEN utility)
files, printing 8-16
files, program 2-8, 3-11, 7-17f, 8-41
(see also save files)
files, random 4-3, 4-9ff, 4-18ff, 5-1, 8-9
files, rename 3-76
files, reserved 3-94f
files, resolution 8-7
files, save 4-1f, 7-1, 8-71
files, save without extension (data file) 4-2
files, scratch 4-2
files, screen 4-2, 5-6, 6-1ff, 6-7
files, sequential 8-9
files, source Preface, 2-2, 2-8, 3-11, 3-22, 3-45, 3-75, 7-11, 7-17f
files, subroutine 4-2
files, table 4-2, 4-10f, 5-1ff, 5-12f
(see also File Maintenance Utility)
files, table (for file maintenance) (FM.TB) 5-2, 5-4ff, 5-12ff
files, table
printing table file list 5-12
(see also FMTABPRINT utility)
files, table, setting up 5-1ff
files, tag 2-8, 4-19ff, 4-25, 7-26
(see also TBUILD)
files, tape 8-13
files, temporary 4-2
files, text 2-8, 3-34f, 3-60f, 7-6ff, 7-11
files, user directory or subpartition (.DR) 4-2
FILL$ 3-25, 4-15ff
fill a string or substring with a value 3-25, 3-70f, 4-15ff
fill character 3-70f
find a key in an index 3-41f
find an available record 4-19ff, 5-3f
(see also GETREC.SL)
find available channel 4-15ff
(see also OPEN utility)
FIND, EDIT command 7-13, 7-19
(see also locating strings, with EDIT utility)
FINDFILE.SL 4-4, 4-12, 7-22
(see also subroutines, subfiles, OPEN utility program)
FIND key (FM template) 5-3f, 5-10
(see also File Maintenance)
FIND NEXT key (FM template) 5-3f, 5-10
(see also File Maintenance)
find subfile, build C1 array 7-22
(see also FINDFILE.SL)
fixed length records 3-74, 3-104f, 4-3
flags, status 4-5ff, 5-5
FLOAD command 8-15
floating decimal 6-4
FM
(see File Maintenance)
FMLOG utility 5-1ff, 5-14
(see also File Maintenance (FM))
FMPRINT utility 5-1ff, 5-14
(see also File Maintenance (FM))
FMSG (OPCLI command) 10-15
FMTABPRINT utility 5-1ff, 5-12
(see also File Maintenance (FM), Table File list, printing)
FM.TB (table file, file maintenance) 5-2
FN prefix 3-18
force a log off 10-15
FOR/NEXT 3-26f, 3-90
FOR/NEXT stack 3-90
foreground 10-3f
form feed terminator 3-36f, 3-66ff, 3-70f
FORM.SL subroutine 6-7, 7-23, 7-44
(see also Screen Handling)
format change 5-4
FORMAT CHANGE key (FM template) 5-3, 5-7, 5-11
(see also File Maintenance)
format code, field 5-9
format for INFOS statements 9-1
FORMAT function key 5-4, 5-7
format, character 2-8, 3-22, 3-34, 3-45, 3-65ff, 3-68f, 7-11, 7-17f
(see also text, listing, source files, EDIT utility)
format, compile 7-11, 8-6
(see also save file)
format control file, editing 8-72
format, dump 8-12
format, output 5-14
(see FM data file, printing)
format, screen 5-6
format types 5-7
formats, field 5-8
formatted fields 6-7
(see also FORM.SL)
formatted fields, protect/unprotect 7-38
(see also PROTFORN.SL)
formatted screen input/output 7-23
(see also FORM.SL subroutine)
formats, page 5-5
formats, PRINT USING 3-70f
formatted input/output, special characters 6-4ff
formatting techniques to print numbers 2-3
FPRINT command 8-16

Licensed Material-Property of Data General Corporation

FREE command 8-17
free a device from exclusive use 8-17
free reserved file 3-94f
full output format 5-14
function key input 7-45
function key templates
FM 5-3
SM 6-2ff
function keys 5-3ff, 6-1ff
function keys on a 6053 terminal 3-88, 5-2, 5-4
function keys, using 5-10f
function number, for function key 7-44
functions 2-2, 2-1, 4-17
functions, arithmetic (or numeric) 2-2, 2-8
functions, string 20-2, 2-67f, 2-8
function, user-defined 3-18

GDIR command 8-17
general format for INFOS statements 9-1
generate CLI commands 7-24, 8-1
(see also GETCM.SL subroutine)
generating PUSH space 10-10
generation and BSG 10-1 to 10-9
generic kyes 9-1
get an available record 4-12ff, 4-19ff, 4-30ff
(see also GETREC.SL)
GETCM.SL subroutine 7-24, 8-1
(see also Read CLI COM file)
get next available record 7-24
(see also GETREC.SL subroutine)
GETREC.SL subroutine 4-4ff, 4-12ff, 4-19ff, 4-30ff, 7-25
(see also next available record, linked-available record, dynamic record allocation)
global switches on BASIC 3-98, 8-1
GOSUB/RETURN 3-28f, 2-2, 3-32, 3-75, 3-89f, 7-1
GOSUB/RETURN stack 3-90
GOTO 3-30, 3-32, 3-75, 3-89
GPOS (byte position in file) 3-31, 3-8, 3-10, 4-29, 4-33f
GSDIR command 8-18
GSYS command 8-18
GTOD command 8-19

half-duplex 3-89
hardware multiply/divide 10-3
\HEAD DOC command 7-7f
(see also DOC utility)
headings, page 7-6, 7-7f, 8-29, 8-37, 8-41
(see also DOC utility, \HEAD DOC command)
headings, section 7-6, 7-7f
(see also DOC utility, \HEAD DOC command)
HELLO program 3-86, 3-90, 10-10f, 10-13f
highest key, INFOS index 9-3
high intensity fields 3-68, 6-1
(see also unprotected fields)
home function key 5-4, 5-10, 6-5
(see also File Maintenance)

.I (keyboard edit command) 1-4
I, EDIT command 7-16f, 7-19
(see also inserting a line with EDIT utility)
IB, EDIT command 7-16f, 7-19
(see also inserting a line with EDIT utility)
IBUILD utility 4-6f, 4-18ff, 4-22ff, 7-26ff
(see also Index Files)
IBUILD error codes 7-27
ID, user 1-1, 5-7
.IDEF start-up sequence 10-1
IF 3-32f, 2-5, 2-7
IKEY 1-3, 3-59, 3-88ff, 7-17, 7-43
IKEY indicator 3-59, 3-89f, 3-98, 5-10
IKEY (OPCLI command) 10-15
illegal blocking factor (167) (IBUILD error code) 7-27
illegal index file offset (-52) (IBUILD error code) 7-27
illegal index record size (165) (IBUILD error code) 7-27
immediate CLI execution 7-3
increment 3-76f
index blocks 3-38ff, 3-41f, 4-7
INDEXBLD, utility 7-28, 4-6f, 4-18ff, 4-20ff, 7-26
(see also Index Files)
INDEXCALC utility 7-29, 4-6f
index entry pointer 7-30
index file, build 7-28, 7-47
(see also INDEXBLD utility, XBUILD utility)
index file, build from sorted tag or data file 7-26f
(see also IBUILD)
index file buffer string 4-35
index file descriptor string
(see descriptor string, index)
index file, initialize 7-31f
(see also INITFILE utility, INITDEX.SL subroutine)
index file, print 7-30
(see also INDEXPRT utility)
index file, rebuild 7-28
(see also INDEXBLD utility)
index file, repack 7-26
(see also blocking factor)
index files 2-8, 3-38ff, 3-41f, 3-61, 3-74, 4-6f, 4-9ff, 4-34ff, 5-1, 5-5, 7-28f, 7-42
index files, creating 4-18ff, 5-1
index files, rebuild 4-18ff, 4-20ff
index files, structure 4-7f
INDEXBLD utility 4-6f, 4-18ff, 4-20ff
(see also index files)
INDEXCALC utility 4-6f, 4-19ff
(see also Index Files)
index information, calculate 7-29
(see also INDEXCALC utility)
INDEXPRT utility 7-30
(see also print an index)
Index Sequential Access Method (ISAM) 3-38ff, 3-41f, 4-6f, 5-1, 5-10, Chapter 9
index space exhausted (68), (IBUILD error code) 7-27

Licensed Material-Property of Data General Corporation

indicator 5-6
(see also File Maintenance)
indirect files (@) 8-1
INFOS interface 1-1, Chapter 9, 10-1, 10-4
INFOS keywords 2-16, Chapter 9
INFOS records Chapter 9
INFOS statement format 9-1ff
INFOS system 1-1, Chapter 9
inhibit execution of News program 10-11
inhibit messages 10-11
INIT command 8-19, 8-31
INITDEX.SL subroutine 7-32, 4-6f, 4-18ff
(see also INDEXBLD, INDEX FILES)
INITFILE utility 7-31f, 4-4, 4-6f, 4-9ff, 4-11ff, 4-19ff, 5-1f
initialize a device 8-19
initialize a variable 3-44, 3-72f
initialize index file 7-31f
(see also INITFILE utility, INITDEX.SL subroutine)
initialize data file 7-31
(see also INITFILE utility)
INPUT 3-34f, 2-3, 2-5f, 7-2
input binary data from file 3-73f
input blocks from file or common area 3-8
input buffer, clear 3-90
input buffer size 10-5
input characters, limit the number of 3-35f, 3-67, 3-88
input data in character format 3-34f, 3-36f
input device, open for access 3-60f
INPUT FILE 3-34f, 2-3, 3-66, 4-28
INPUT FILE USING 3-36f, 3-66, 4-28
input file, alternate 7-12
(see also EDIT utility)
input file, primary 7-12
(see also EDIT utility)
input in a specified time 3-101
input to screen file 7-23
(see also FORM.SL)
input/output access modes 3-60f
input/output, file 2-8, 3-8, 3-34f, 3-36f, 3-60f, 3-73f
input/output, file, for EDIT utility 7-12
input/output, formatted screen 7-23
(see also FORM.SL)
input/output, formatted - special chars. 6-4ff
input/output, mag tape and cassette 3-52ff, 3-60f
input/output files, specifying for EDIT 7-12
Input/Output variables 4-26ff
INPUT query prompt 2-5, 3-34f, 3-36f
input record out of sequence, (166) (IBUILD error code) 7-27
INPUT time remaining 3-98
INPUT USING 3-36f
INSERT CHARS (SHIFT-F1), SM function key 6-2f, 6-6
insert mode 7-11ff, 7-14f, 7-17
(see also EDIT utility, interrupt key)
INSERT NEW LINE (SM function key) 6-2f, 6-6
inserting a line with EDIT utility 7-16f
(see also I, EDIT command, IB, EDIT command)
installation - DOS 10-2
installation - RDOS 10-1
INT (integer function) 3-38
integer field (binary) single precision (I) 5-8
(see also field formats)
integer key type 7-30
(see also key types)
integer, single precision 7-80
(see also percent (%) sign in variable name)
integer, triple-precision 7-40
(see also number sign in variable name)
integer, unsigned conversion 7-46
integers 2-3, 3-38, 3-85
intensity, low/high 3-68
interactive priority 10-16
interrupt, disable 3-59, 3-88ff
interprogram communications (common area) files (.CM) 4-2
interrupt, enable 3-59, 3-88ff, 10-10
interrupt key 1-3, 1-6, 3-59, 3-67, 3-88ff, 7-17, 7-43
interrupt I/O 3-104f
interrupting programs 1-6, 3-59, 3-88ff
INT, ON IKEY THEN 3-59
I/O bound time slice 10-15
I/O priority 10-11
I/O status indicators 7-36
(see also PORTS utility)
I/O usage 3-98
ISAM (Indexed Sequential Access Method) 3-38ff, 3-41f, 4-6f, 5-1 5-10, Chapter 9
ISAM access 4-18ff, 4-28ff, 4-34ff, Chapter 9
(see also Index Files, Utilities)
ISAM file structure 1-1, 4-6f
ISLICE (OPCLI command) 10-15

.JB extension 10-14
job, de-attached 3-94, 3-98, 7-1, 7-36, 10-11
job execution, automatic 10-14
job number 3-98, 7-36, 10-15
jobs, maximum number 10-5
julian date 3-91f
justifying text 7-6ff
(see also DOC utility, \STX command \ETX command)

K (print format) 3-70f
KADD 3-38f, 4-6f, 4-18ff, 4-25, 4-35, 7-26, 7-28, 7-47
KDEL 3-40f, 4-6f, 4-18ff, 4-36
(see also File Access)
key 4-6f, 7-42, Chapter 9
(see also index files)
key, add to indexfile 3-38f, 4-6f, 4-18ff, 4-25
(see also File Access)
key, alphanumeric unsigned 7-39

Licensed Material-Property of Data General Corporation

key, character-delete 3-67, 3-87, 7-43
key compression 9-1
(see also crammed keys)
key, de-attached 7-2, 7-43
(see also ATTACH utility)
key, delete from index 3-40ff, 4-6f, 4-18ff, 4-36
(see also File Access)
key descrptor records 5-5
key entry 3-38ff, 3-41f, 4-7, 4-35, 5-10
key, fields 4-22f, 4-25, 5-6, 7-21, 7-26, 7-42
key, find in index 3-41f, 4-6f, 4-18ff, 4-37f
(see also File Access)
key, interrupt
(see IKEY)
key, line cancel 3-67, 3-87, 7-43
key, next in sequence 3-42, 4-6f, 4-18ff, 4-38f
(see also File Access)
key string, index 3-38ff, 3-41f, 4-35
key type 7-30
keyboard editing commands 1-4, 3-45
keyboard, lock 7-44
keyboard mode 1-2ff, 1-1, 2-1, 7-3, 8-28
keyboard, terminal Preface
keys, concatenated 4-7, 4-25
keys, duplicate 4-6f, 4-19ff, 4-35f, 7-31f, 9-1
(see also Index Files)
keys, function 3-88, 5-2, 5-4, 6-1ff
keys, generic 9-1
keys, INFOS Chapter 9
Keys, multiple 7-21, 9-1
keys per block 4-6f, 7-29
(see also INDEXCALC utility)
keyword 2-1, Preface, 2-8ff
keyword/format description Appendix B
keyword summary Appendix B
KFIND 3-41f, 4-6F, 4-18ff, 4-37
(see also File Access)
KILL 10-16, 10-21f
KNEXT 3-42, 4-6f, 4-18ff, 4-38f
(see also File Access)

L, EDIT command 7-13, 7-19
(see also locating strings with EDIT utility)
L7 (print format) 3-70f
L8 (print format) 3-70f
L9 (print format) 3-70f
L10 (print format) 3-70f
L11 (print format) 3-70f
labeling of GOTOs and GOSUBs 3-75, 3-89
layout, screen file 6-14
LC, EDIT command 7-13, 7-19
(see also locating strings, with EDIT utility)
leading characters 3-70
left-justified field 7-44
left page margins 7-7ff
(see also \MAR(DOC) command, DOC utility)
LEN 3-43, 2-6f
length of a string 3-43
length, program and data 3-98
LET 3-44, 1-6f, 2-1, 2-3, 2-6
level 0 4-7
$LIB (library directory) 4-10, 8-20, 10-14
LIBRARY command 8-20
library directory 3-12, 3-22, 3-46, 3-81, 3-97, 4-10, 8-20
limit number of input characters 3-35
line cancel echo 3-67, 3-89
line cancel key 3-67, 3-87, 7-43
line centering 7-6
(see DOC utility)
line current 7-11ff, 7-13f, 7-17
(see also prompt, 000:)
line feed 1-3, 7-7f
LINE FEED key 1-3
line increment 7-15
line, inserting with EDIT utility 7-16f
(see also I, EDIT command, IB, EDIT command)
line numbers 1-2ff, 1-5ff, 2-1, 3-45, 3-75ff, 3-81
line numbers, renumber 3-76f
line numbers, subroutine 2-2, 3-75
line printer ($LPT) 4-1f, 5-12, 7-6
line printer listings Preface, 3-45, 8-29
line speed, default 10-4
lines, copying with EDIT utility 7-15f
(see also DITTO, EDIT command)
lines, deleting with EDIT utility 7-16
(see also D, EDIT command)
LINES n, EDIT command 7-19
lines, moving with EDIT utility 7-15f
(see also MOVE, EDIT command)
lines, replacing with EDIT utility 7-16
(see also R, EDIT command)
LINE$, screen literals 6-14, 7-44
(see also Screen File layout)
link alternate name to file 8-21
(see also LINK)
link access 8-8
LINK command 8-21
link entries, directory 4-2
link entries, removing (UNLINK) 8-41
link entry 3-76, 8-21
link entry file attribute 4-3,
linked-available-record 4-4, 4-28ff, 5-5, 7-25, 7-31, 7-40, 7-42, 7-47
(see also GETREC.SL)
linked-available-record access 4-30ff, 5-1
linked-available-record chain 4-30
linking, no (attribute) 8-6
link to a subindex (INFOS) 9-3
lines in EDIT disk buffer, maximum number 7-12
(see also LINESn, EDIT command)
LIST 3-45f, 1-3ff, 2-1, 2-8, 3-22, 4-3, 8-22
list character contents of program (source statements) 3-45f, 3-75
list variables in program 7-47
(see also VAR utility)
listings, line printer Preface, 3-45,
listing (.LS) 4-2, 8-37, 8-41

Licensed Material-Property of Data General Corporation

listing files Preface, 2-8, 3-11, 3-22, 3-45, 3-75, 7-1 7-11, 7-17f, 8-22, 8-25f, 10-2
listing, load map 10-2
LNKSI, NFOS 9-3
LOAD 3-46, 1-3, 2-1, 3-81f, 8-24
load map listing 10-2
loading from diskettes 10-1
loading from magnetic tape 10-1
local switches 8-1
locating strings, with EDIT utility 7-13
(see also L, EDIT command, LC, EDIT command, FIND, EDIT command)
location counter 8-16
lock area 3-47f, 4-39, 10-5
(see also File Locking, LOCK/UNLOCK)
lock flag, automatic 3-38ff, 3-41f, 7-32
lock keyboard 7-44
lock identifier 3-47f, 4-39, 7-33
(see also LOCKS utility)
lock table, index file 3-47f, 4-39
(see also File Locking)
LOCK/UNLOCK 3-47f, 3-39f, 3-41f, 4-39
lock, unlock a key entry (INFOS) 9-2
locks, display current 7-33
(see also LOCKS utility)
LOCKS utility 7-33, 3-47f, 4-39
log file 5-1, 5-7, 5-14
logging, by FM (File Maintenance) 5-7
logging off 1-7, 3-11, 7-2
logging on 1-1, 1-7, 3-94,
logical AND of expressions 3-5, 3-84,
logical bit shift 3-84
logical filename 3-39, 3-47f, 4-39, 5-5, 7-22, 7-30f 7-35, 7-40
logical files 4-4, 4-12ff, 4-15ff, 4-30ff, 4-32, 7-4 7-37
(see also physical files, subfiles)
logical inclusive OR 3-62f, 3-84
log-off, automatic 10-11
log-on message, automatic 10-14
log on time 3-98
loop, FOR/NEXT 3-26f
low intensity fields 3-68, 6-1
(see also protected fields)

lowercase characters 3-89, 3-93, 6-4, 7-13
(see also EDIT utility, ACL, EDIT command, DLC, EDIT command)
$LPT (line printer) 3-45f, 3-61, 4-1f, 7-6, 8-29, 8-37
.LS extension 7-17
LSPEED utility 7-33
(see also set terminal's speed)
LSTCOM command 8-25
LSTMERGE 8-26

magnetic tape, I/O 3-52ff, 3-60f, 8-14, 8-38
magnetic tape, loading from 10-1
maintain files, keywords 2-11ff
Maintenance, File 2-2
Maintenance, Screen 2-2
manager, system 1-1, 10-1
mapped NOVA3 10-3
\MAR (DOC command) 7-6ff
(see also DOC utility)
margins, left page 7-7ff
margins, right page 7-7ff
margins, setting 7-7ff
(see also \MAR DOC command, DOC utility)
master directory 8-27
master processor 10-4
MAX 3-49
MAXn, EDIT command 7-19
max fields 5-6
(see also File Maintenance)
maximum characters allowed on input 3-67, 3-88
maximum characters per line, for EDIT disk buffer 7-11
(see also MAXn EDIT command)
maximum columns per line 3-64, 3-88
maximum number of bytes in a string 2-6, 3-20f
maximum number of jobs 10-5
maximum number of lines, in EDIT disk buffer 7-12
(see also LINESn EDIT command)
maximum number of records 7-29, 7-31, 7-40
(see index file)
maximum of two expressions 3-49
maximum program and data length 3-98
maximum program size 10-5
MDIR command 8-27
menu concept 10-13
menu program 10-12
merge program source statements 3-22
merge listing files 8-26
merge program statements 7-1ff
merge subroutines 7-1ff
message, automatic log-on 10-14
messages, inhibit 3-51, 3-89, 10-11, 10-15
messages to terminals 3-51, 3-89, 10-15
MICRONOVA1 10-3
MIN 3-49
minimum node size, INFOS index 9-2
minimum of two expressions 3-49
minus sign (-) 3-70f, 7-17, 7-46
(see also inserting a line with EDIT utility, plus sign (+))
miscellaneous keywords 2-13ff
MOD 3-50, 3-91
mode, access for files 3-60f, 7-35
mode, command (for EDIT utility) 7-11ff
mode, brief (EDIT utility) 7-18
(see also BR, EDIT command)
mode, comment 7-11, 7-17f
(see also EDIT utility, COM, EDIT command)
mode, insert 7-11ff, 7-14f, 7-17
(see also EDIT utility, interrupt key)
mode, keyboard 1-2ff, 1-1
mode numbers 4-15ff

Licensed Material-Property of Data General Corporation

mode, program 1-1
mode, Verify (EDIT utility) 7-18, 7-20
(see also V, EDIT command)
mode 7 3-52ff, 3-60f
modem control 10-3f
modes 0 to 7 3-60f
month of year 3-98
motion, relative (INFOS) Chapter 9
MOVE command (CLI) 8-27
MOVE, EDIT command 7-15f, 7-20
(see also moving lines with EDIT utility)
move cursor 3-68
moving lines with EDIT utility 7-15f
(see also MOVE, EDIT command)
MOVETABREC utility 5-14
(see also table file records)
MSG 3-51
MSG (OPCLI command) 10-15
MTDIO 3-52ff, 3-60f, 3-62f, 3-84
MUL.SL subroutine 7-34
(see also calculate percent of a number)
multiple key fields 7-42
multiple keys 7-21, 9-1
multiple level indexing 9-1
multiplexor line, ALM 7-33, 10-1, 10-4
(see also LSPEED utility, baud rate)
multiplication 2-5

N, EDIT command 7-14, 7-20
Nw.d 5-9
(see also field format code, digit subfield)
names, resreved variable 2-2
names, variable 2-2f
negative number 2-3, 3-4, 3-83
nested parentheses 2-5
nesting loops 3-26f, 3-28f, 3-90
nesting user-defined functions 3-18
NEW 3-56, 1-3, 2-1, 3-81
NEW LINE key Preface
NEWS program 10-14
NEWS program, inhibit execution 10-11, 10-14
NEWS.DS file 10-14
NEWS.LG file 10-14
NEXT 3-26f, 5-4
next available channel 7-22
next available record 4-4, 4-19ff, 7-4, 7-24
next field 5-4
next key in sequence 3-42
NFOS CLOSE 9-2
NFOS DFSI 9-2
NFOS DELETE 9-2
NFOS DELSI 9-3
NFOS keyword (INFOS statements) Chapter 9
NFOS LNKSI 9-3
NFOS OPEN 9-3
NFOS READ 9-3
NFOS REINS 9-3
NFOS RETDF 9-3
NFOS RETHK 9-3
NFOS RETKY 9-3f
NFOS RETST 9-4
NFOS REWRITE 9-4
NFOS WRITE 9-4
\NLF, DOC command 7-7ff
(see also DOC utility, overprinting)
node size, index (INFOS) Chapter 9
no echo 3-89
no - IKEY flag 3-59, 3-89
no line feed 7-7f
(see also DOC utility, \NLF DOC command, overprinting)
no message bit 3-51, 3-89, 10-15
NOVA2, unmapped 10-3
NOVA3, mapped 10-3
null string 2-6, 3-25, 3-70
null terminator 3-36f
nulls, fill file with 7-31
(see also INITFILE utility)
number, binary 2-3, 3-7
number, decimal 2-3
number, extract from string 2-6, 3-7, 3-12, 3-83
number, make an integer 3-38
number, negative 2-3, 3-4, 3-83
number, octal 2-3
number of blocks at level 0 7-29
number of printed lines per page 7-6
(see also DOC utility)
number of sectors in data file 7-29
(see also blocks)
number of sectors in index 7-29
(see also blocks)
number, put in string 3-12, 3-44
number sign in variable name 2-3, 7-40
number, random 3-72, 3-80
numbers 2-3ff
numbers, format techniques for printing 2-3
numbers, unique 3-98
numeric expressions 2-5, 2-2, 3-44, 3-49, 3-83
numeric expression, sign of 3-83
numeric functions 2-2, 2-8
numeric, signed 6-4, 7-46
(see also data types for input/output)
numeric variable, double precision 2-3ff, 3-44, 3-73f, 3-104f
numeric variables 2-3ff, 3-44, 3-73f, 3-104f
numeric variable, single precision 2-3ff, 3-44, 3-73f, 3-104f
numeric variable, triple precision 2-3ff, 3-44, 3-73f, 3-104f

O (print format) 3-70f
O, EDIT command 7-12, 7-20
(see also file input/output, EDIT, O, EDIT command)
octal format 3-70, 8-16
octal number 2-3, 3-70
OMIT, EDIT command 7-18, 7-20
(see also stopping an EDIT session)

Licensed Material-Property of Data General Corporation

ON ERR 3-56f, 1-7, 3-24
ON GOTO and ON GOSUB 3-57f
ON IKEY 3-59, 1-7, 3-88f
one dimensional arrays 2-4f, 3-20f
OP account 10-11, 10-15
(see also System Security)
OPCLI commands 10-15
OPCLI utility 10-15
OPEN FILE 3-60f, 4-9ff
OPEN modes 4-16
(see also exclusive, shared)
OPEN, NFOS 9-3
open physical files, subfiles 7-35
(see OPEN utility)
OPEN utility 7-35, 4-4, 4-9ff, 4-12, 4-15ff
(see also subfiles, find available channel, FINDFILE.SL)
operating systems, RDOS and DOS 1-1, 8-18, 10-1
operator, concatenation (comma) 2-6
operator, system 1-1, 3-51, 10-1, 10-15
operators, arithmetic and relational 2-5
OR, logical inclusive 3-62f, 3-84
order, ascending, descending 7-21, 7-39
output binary data to file 3-104f
output blocks to file or common area 3-10
output buffer size 10-5
output characters to file or device 3-65ff, 3-68f, 3-70f
output device, access to 3-60f
output device, default 3-90
output formats FM data file 5-14
output in octal format 3-70
output right justified 3-70f
output to field 7-38
(see also PROTFORM.SL subroutine)
output to screen file 7-23
(see also FORM.SL)
overlay file (.OL) 4-2, 8-40
overlays, default 10-6
overlays, virtual 10-6
overprinting 7-7f
(see also DOC utility, \NLF DOC command)
overwrite character 3-70f
OW, EDIT command 7-12, 7-20
(see also File Input/Output, EDIT, page mark, O, EDIT command)

.P (keyboard edit command) 1-4
P (print format) 3-70f
P, EDIT command 7-20
P!, EDIT command 7-20
packed output format 5-14
packet definitions, INFOS 9-4
pad characters 3-67, 3-89
PAGE 3-64, 3-66, 3-100, 5-7f
PAGE CHANGE key
(see CHANGE PAGE key)
page change 5-4
page, default 5-6
page, display 5-6
\PAGE, DOC command 7-7f
PAGE, EDIT command 7-12, 7-20
(see also file input/output, EDIT, A, EDIT command, O, EDIT command)
page formats 5-5
page headings or text files input files 7-6, 7-7f
(see also DOC utility, \HEAD DOC command)
page lengths of text input file 7-6ff
(see also DOC utility)
page numbers, setting 7-8
(see also DOC utility, \SPAGE DOC command)
page width of a job 3-64, 3-66f, 3-88, 3-100
paging of text input files 7-6f
(see also DOC utility)
paging, conditional 7-6ff
(see also DOC utility, \CPAGE command)
paging, unconditional 7-7
(see alos DOC utility, \PAGE command)
PARAM entry 7-35, 7-40
(see also OPEN utility)
PARAM file 4-4, 4-9ff, 4-11, 4-2ff, 5-2, 7-22, 7-28ff, 7-40
parentheses in expressions 2-5
parentheses, nested 2-5
parenthesis, for command replication 8-1
parity, tape 3-52ff
partial key, approximate match 4-7
(see also Index File)
partial records, DBAM 9-1ff
partition, deleting 8-10
partition size (1K word blocks) 10-3
partitioning, disk 4-2, 4-3, 8-11
partitions 4-2, 8-5, 8-8, 8-11, 8-19, 8-27
pass information between programs 3-8, 3-10, 3-12, 3-86
password 1-1, 5-1, 10-11
password - account system 10-10ff
password, for data file 5-7
PCNT (percentage) 7-5, 7-34
(see also DIV.SL, MUL.SL)
percent of a number, calculate 7-34
(see also MUL.SL)
percent sign in variable name 2-3, 7-40
percentage (PCNT) 7-5, 7-34
(see also DIV.SL, MUL.SL)
perform calculations 2-3, 3-44, 3-65
(period) . (keyboard edit command) 1-4
permanent file (P), file attribute 4-3, 8-6
physical file 3-23, 4-4, 4-12ff, 4-15ff, 7-41, 7-43
(see also logical subfiles)

Licensed Material-Property of Data General Corporation

physical files, open 7-35
(see also OPEN utility)
plus sign (+), with EDIT 7-17, 7-46
(see also inserting a line with EDIT utility, minus (-) sign)
pointer, data list 3-79
pointer, file or byte 3-8, 3-10, 3-31, 3-61, 3-64f, 3-73f, 3-104f
pointer, record 3-38ff, 3-41f, 4-9, 4-17ff, 4-19ff, 4-25, 4-36f, 4-38f, 7-42
POP CLI command 8-28, 7-3
port number 3-98, 8-32
port, send/receive messages 3-51
PORTS utility 7-36
POSFL.SL 7-37, 3-64f, 4-4, 4-12ff, 4-19ff, 4-30ff, 7-25
position, byte (POSFL.SL) 4-4
(see also file characteristics, record position)
POSITION FILE 3-64f, 4-29ff
position file pointer to record 7-37
(see also POSFL.SL subroutine)
positioning, cursor 3-34f, 3-36f, 3-65ff, 3-68f, 7-45
positioning specification (INFOS) Chapter 9
position of bytes in string and substring 2-6f
position of byte in file 3-8, 3-10, 3-31, 3-61, 3-64f
position, record (POSFL.SL) 4-4, 7-37
(see also file characteristics, byte position)
position to byte
(see POSITION FILE, see also POSFL.SL)
position to record 4-12ff, 4-19ff, 4-30ff
(see also POSFL.SL)
positive sign 3-83
powers of two table 3-6
precision, double (0) 10-3
precision, double numeric variable 2-3f
precision, single numeric variable 2-3f
precision, triple 10-3
precision, triple numeric variable 2-3f
previous field 5-4
prewritten subroutine 2-2
PRI (OPCLI command) 10-16
primary interrupt key 3-67, 3-88
primary partition 8-8
primary unpend key 3-67, 3-87f
PRINT 3-65ff, 1-3ff, 1-5f, 2-1, 2-5ff, 3-34, 3-68f, 8-29
PRINT FILE 3-65ff, 2-3, 3-34, 3-68f, 4-28f
PRINT FILE USING 3-70f, 4-28f
print index file 7-30
(see also INDEXPRT utility)
print series of text input files (DOC) 7-6ff, 8-29
(see also Subroutine and Utilities)
PRINT SM command 6-6
print table of contents 7-11
(see also DOCTOC utility)
printed lines per page, number 7-6f
(see also DOC utility)
printing numbers, formatting techniques 2-3, 3-66, 3-70f
PRINT USING 3-70f, 2-3, 5-9
print tab zones 3-66, 3-100
primary input file 7-12
(see also EDIT utility)
priority, compute 10-11, 10-16
priority, disk 10-11
priority, I/O 10-11, 10-16
priority, system call 10-16
privileged system calls 10-16 to 10-21
PROCESS CMND (F11), SM function key 6-2f, 6-6
Processor 10-4
processor number 3-98
processor system, dual 10-4
(see also slave processor)
program 1-2, 2-8, 3-81
program, commenting a 3-75
program, comments to 4-2
program, continue execution of 3-15, 3-81
program, current name 3-90
program development keywords 2-9f
program, editing 1-3ff, 3-45
program, erasing statements 3-23
program, error trap 3-56f
program, execution 8-6
program file (save file) 2-8, 3-11, 3-22, 3-46, 3-76, 3-78, 3-81f, 3-96f, 7-1, 7-17f
program in character format 7-11, 7-17f
(see also EDIT utility)
program, input to 3-34f
program, interrupt trap 3-59
program length 3-98
program, listing 3-45
program, loading 3-46, 3-81
program loop 3-26f, 3-30
program menu 10-12f
program mode 1-1
program, renumber 3-76f
program, replace 3-78
program, RUN-only 3-56, 3-59
programs, running and interrupting 1-6, 3-81
program, saving 3-46, 3-82
program size, maximum 10-5
program, size of 3-85
program source statements 3-22, 3-45
program statements, merge 7-1ff
program, stop execution 3-96
program terminator 3-21
program, trap an error 3-56f
program, trap an interrupt 3-59, 3-88ff
program variables, rename 7-40
(see also RNAM utility)
program, writing 1-3
programmer, system 1-1
programs, source
(see source files)

Licensed Material-Property of Data General Corporation

prompt, asterisk 1-1ff, 1-7
prompt, BASIC CLI 1-7
prompt, field 5-8
prompt, for EDIT utility 7-11ff
(see also 000: prompt, EDIT utility)
prompt, for INPUT query 2-5, 3-34f, 3-36f
prompt, OPCLI 10-15
prompt, RDOS/DOS CLI (R) 10-10
PROTECT TO EOL (F2), SM function key 6-2f, 6-5
protect/unprotect formatted screen field 7-38
(see also PROTFORM.SL subroutine)
protected fields 6-1, 7-28, 7-44
PROTFORM.SL subroutine 7-38, 6-7
(see also Screen Handling)
PRTCOM command 8-29, 8-3
PRTCOM, build a documentation file for 8-3
(see BLDCOM)
pseudo-random integer 3-80
punctuation, PRINT 3-65ff, 3-68f
push level 3-98
push space, generating 10-10
put a number into a string 3-7, 3-12

\QC DOC command 7-6ff
(see also DOC utility)
QFILESORT utility 7-39, 4-22
question mark (?) prompt 3-34f, 3-36f
QUICKILL 10-21f
QUIT command 8-30
quotation marks, string literal 2-5

R, EDIT command 7-16, 7-20
(see also replacing lines with EDIT utility)
R (print format) 3-70f
random file access 3-31, 3-60f, 3-64f, 3-73f, 3-104f, 4-16, 4-28ff, 4-29f, 5-1, 5-5
random file organization (R) 4-3ff
random files 4-3, 4-9ff, 4-18ff, 5-1, 8-9
(see also file attributes)
random number generator 3-72, 3-80
random numbers 3-73, 3-80
RANDOMIZE 3-72, 3-80
range of lines 3-45
RDOS 1-1, 8-39
RDOS/DOS BSG program 10-2
RDOS/DOS CLI 2-2, 8-1
RDOS/DOC CLI commands 2-2
RDOS/DOS CLI prompt (R) 10-10
RDOS/DOS system calls 10-18
RDOS/DOS tape format 3-53f
RDOS dump format 10-1
RDOS - INSTALLATION 10-1
RDOS INFOS system Chapter 9, 10-1
READ 3-72f, 2-3, 2-6, 3-17, 3-79
read CLI COM file 7-24
(see also GETCM.SL subroutine)
read INFOS record/partial record 9-3
read blocks from file 3-8, 3-10
READ FILE 2-3, 3-73f, 4-29ff
READ, NFOS 9-3
read protected file attribute 4-3, 8-6
read records to a file 4-29ff
(see also READ FILE)
READ, SM command 6-6
read tape records 3-52ff
Real-time Disk Operating System 1-1
rebuild index file 7-28
(see also INDEXBLD utility)
record, allocate a new 4-32
record chain, deleted 4-5
(see also Record 0)
record, control 5-4ff
(see also File Maintenance)
record, definition 4-3ff
(see also binary string)
record, delete 4-4
(see also DELREC.SL)
record, dynamic allocation 4-4, 5-1, 7-4, 7-25, 7-47
(see also GETREC.SL)
record in lock area 3-47f
record, linked - available 4-4, 4-28ff, 5-1, 5-5, 7-25, 7-42, 7-47
(see also GETREC.SL)
record lock/unlock 3-47f
record, next available 4-4, 4-19ff, 7-4
(see also GETREC.SL)
record number, data file 3-38ff, 3-41f, 3-64f, 3-104f, 4-4, 4-12f, 5-10, 7-31, 7-37, 7-42
record pointer to data file 3-38ff, 3-41f, 4-36f, 4-38f, 7-26, 7-42
record position (POSFL.SL) 4-4
(see also file characteristics, byte position)
record, rewriting 3-104f, 4-30
record size, data file 3-64f, 3-74, 3-104f, 4-12ff, 7-31, 7-40
record status 4-32, 5-1, 5-11, 7-4, 7-25, 7-40, 7-47, 9-4
record, tag file 7-26
record types 5-1f, 5-5, 5-7, 5-11
record 0 4-4ff, 4-30ff
(see also DELREC.SL, GETREC.SL, record number, INITFILE)
record 0, build 7-31
(see also INITFILE utility)
records, data file 5-6
records, field descriptor 5-5, 5-12ff
records, key descriptor 5-5
records, maximum number 7-29, 7-31, 7-40
(see also index files)
records, table file 5-4ff, 5-14
records, tape 3-52ff
records, type 1 5-4ff
records, type 2 5-6f
records, type 3 5-7
records, type 4 5-6ff
records, type 5 5-8f
records, type 6 5-10
redimension an array 2-5

Licensed Material-Property of Data General Corporation

refer to array element 2-4f
reinstate INFOS deleted record 9-3
relational expressions 2-5, 3-32f
relational expressions, strings in 2-7, 3-32f
relational operators 2-5, 3-32f
relative byte within file 3-64f
relative motion (INFOS) Chapter 9
RELEASE command 8-31
RELINK utility 7-40
(see also clean up linked data file)
REM 3-75, 3-89
remainder after dividing two expressions 3-50
remark statement 3-75
remove a file from a directory 3-19
remove existing attributes 8-6
RENAME 8-31, 3-76
rename a device 8-15
rename file's directory entry 3-76
rename (save file's) program variables 7-40
(see also RNAM utility)
RENUMBER 3-76f
repack an index file 7-26
(see also blocking factor)
repeat count 3-70f
REPLACE 3-78, 2-1, 3-46, 3-76, 3-82
replace a program 3-78
replacing lines with EDIT utility 7-16
(see also R, EDIT command)
reseed random number generator 3-72
reserved file or device 3-94f, 8-3, 8-17, 10-6
reserved file table 10-6
reserved variable names 2-2
reset data list pointer 3-79
reset terminal type characteristics 3-67
resolution files 8-7, 8-10, 8-21f, 8-41
RESTORE 3-79, 3-17, 3-72f
restricting access to directories 10-13
retain existing attributes 8-6
retaining values in a stopped program 3-15
RETDF, NFOS 9-3
RETHK, NFOS 9-3
RETKY, NFOS 9-3f
retrieve a key (INFOS) 9-3
retrieve a SAVEd program 3-46
RETST, NFOS 9-4
RETURN 3-28f, 2-2, 3-58
RETURN key
(see carriage return)
return system information 3-98
rewrite an INFOS record 9-4
REWRITE, NFOS 9-4
rewriting a record 3-104f
right-angle prompt, OPCLI 10-15
right-justified fields 5-8
right-justified output 3-70f
right-justify character 6-4
right page margin 7-7ff
(see also \MAC (DOC) command, DOC utility)
RLDR, command line 10-2
RNAM utility 7-40
(see also rename program variables)
RND 3-80, 3-72
Roll-mode (File Maintenance (FM)) 5-2, 5-9, 5-11ff
(see also File Maintenance (FM))
row and column of array 2-4f, 3-20f
RUBOUT key 1-3, 3-87
RUN 3-81, 1-3, 2-1
RUN-only programs 3-56, 3-59, 10-4, 10-13
running and interrupting programs 1-6

S (print format) 3-70f
sample dialog, BSG 10-3 to 10-9
sample output routine, SM 6-11ff
sample program, SM 6-9ff
SAVE 3-82, 1-3, 2-1, 2-8, 3-46, 3-78, 3-81
save file (core image file, program file) 3-11, 3-46, 3-75f, 3-78, 3-81, 3-82, 3-96f, 4-2, 7-11, 8-6
save file attribute 4-3, 8-6
save file, symbol table 7-47
save file without extension 4-2
save program on disk 3-82
SCALE TO EOL (F3), SM function key 6-2ff, 6-6
SCHANS utility 7-41
(see also display system channel assignments)
Scratch file (.SC) 4-2
"SCRATCH" filename 3-78
screen, defining 6-1ff
(see also Screen Maintenance (SM))
screen file layout 6-14
screen filename 5-6
screen file, input to 7-23, 7-44
(see also FORM.SL, UNFORM.SL)
screen file, output to 7-23, 7-44
(see also FORM.SL, UNFORM.SL)
screen files (.Sn) 4-2, 5-6, 6-1ff, 6-7
screen format 5-6
screen input/output, formatted 7-23
(see also FORM.SL)
screen input/output, unformatted 7-44
(see also UNFORM.SL)
Screen Handling Chapter 6
(see Screen Maintenance)
screen, last line read (XLROW) 7-44
Screen Maintenance (SM) 2-2, 5-6, Chapter 6, 7-44
Screen Maintenance (SM) function keys 6-2f
screen literals, LINE$ 6-14
(see also Screen File layout)
screen number 5-6, 6-6, 7-44
screen overflow 7-45
screen, using 6-7ff
SCRN$ 6-14, 7-44
(see also Screen File layout)
SDIR command 8-32
second TTY 10-3
second TTY support 10-1
secondary interrupt key 3-67, 3-88, 3-98
secondary partition, create 8-8
secondary single processor 10-4

Licensed Material-Property of Data General Corporation

secondary unpend key 3-67, 3-88, 3-98
section headings 7-6, 7-7f
(see also DOC utility, \HEAD DOC command)
sections document, produced by DOC utility 7-6
sectors
(see blocks)
sectors, number of, in index 7-29
(see also blocks)
semicolon 2-1, 3-35, 3-66ff, 3-71
semicolons with BASIC CLI commands 1-7, 8-1
send a message to another terminal 3-51, 10-15
sequential file access 3-31, 3-34, 3-60f, 3-73f, 3-104f, 4-16, 4-28ff
sequential file organization 4-3
sequential files 4-3, 4-9ff
(see also variable length records and file attributes)
series of unique numbers 3-98
SET, EDIT command 7-13, 7-20
(see also setting tabs with EDIT)
set block mode 7-44
set page width for a job 3-64
set STMAs 7-44
set terminal's speed 7-33
(see also LSPEED utility)
setting page numbers 7-8
(see also DOC utility, \SPAGE DOC command)
setting margins 7-7ff
(see also \MAR DOC command, DOC utility)
setting tabs, with EDIT 7-13
(see also SET, EDIT command)
SGN 3-83
shared file access 3-60f, 4-16
SHFT 3-84
shift, logical bit 3-84
SHOW (OPCLI command) 10-15
shutting down BASIC 10-21f
sign of an expression (positive/negative) 3-83
sign of numeric after I/O transfer, correct 3-63, 3-74, 3-83
signed numeric 6-4, 7-46
single precision binary representation 2-3f
signed numeric 6-4
(see also data types for input/output)
single precision integer 7-40
(see also percent (%) sign in variable name)
single precision integer (binary) field (I) 5-8
(see also field formats)
single precision variable (%) 4-6
(see also byte tranfers)
single precision numeric variable 2-3ff, 3-44, 3-73f, 3-104f
single processor 10-4
SIZE 3-84
size of program in working storage 3-85
skip, EDIT command 7-12, 7-20
.SL extension 2-2, 7-1
(see also Subroutines and Utilities)
slave processor 10-4
SLINE command 8-32
SM
(see Screen Maintenance)
SM error messages 6-8
SM field definitions 6-14
(see also Screen File layout)
SM function key template 6-3
SM sample output routine 6-11ff
SM sample program 6-9ff
.Sn extension 6-2
(see also Screen Maintenance (SM))
sort disk file 7-21
(see also FILESORT utility)
source file (.SR, BASIC) 4-1f, 8-29
source files (ASCII) Preface, 2-02, 2-8, 3-11, 3-22, 3-45, 3-75, 7-11, 7-17f
source statements (ASCII) 3-22, 3-45, 3-75
space Preface
spaced output format 5-14
\SPAGE DOC command 7-8
(see also DOC utility, setting page numbers)
SPDIS command 8-33
SPEBL command 8-33
special character in variable names 2-3
special characters in crammed strings 3-16, 3-90, 3-103
specific subroutine variables 2-2
specifying input/outut files for EDIT 7-12
(see also FN command)
SPKILL command 8-34
spooling 8-33ff
square root function (SQR) 2-1, 3-85
SQR 3-85, 2-1
.SR extension 7-17
stacks 10-1
stacks, FOR/NEXT and GOSUB/RETURN 3-90
START 8-35, 10-11
start de-attached jobs 10-11
(see also START, CLI command)
START PROTECT (F6), SM function key 6-2f, 6-6f
start random number generator 3-72
start up sequence, .IDEF 10-1
starting block number in file 3-8, 3-10
starting byte of lock area 3-47f
STARTUP program 10-14
STAT command 7-36, 8-35, 10-16
statement number, last error or IKEY 3-98
statements 2-1, 1-2ff, 1-5ff
statements, BLOCK WRITE 4-15ff
statements, erasing 3-23
statements, file access 2-3
statements, renumber 3-76f
status, file 7-40
status flags 4-5f, 5-5
(see also Record 0)
status of IKEY 3-98
status, of record 4-32, 5-1, 5-11, 7-4, 7-25, 7-40, 7-47, 9-4
status values, tape 3-52ff
status word, tape 3-54f, 3-62f, 3-84
status word, user 3-98

Licensed Material-Property of Data General Corporation

STEP 3-26f, 3-76f
STMA, setting 7-44
STMA 1-3, 3-86f
STMA 2-1, 4-25, 7-27
STMA 4 3-87ff, 3-67f
STMA 4,0 7-1
(see ATTACH utility)
STMA 4,6 3-88, 3-59
STMA 4,8 3-88, 3-64
STMA 4,10 3-88, 3-35
STMA 5 3-89f
STMA 6 3-89f
STMA 6,3 3-89, 3-67
STMA 6,4 3-89, 3-51
STMA 6,5 3-89, 3-59
STMA 6,6 3-89, 3-75
STMA 7 3-89f
STMA 7,5 3-89, 3-59
STMA 8 3-90
STMA 8,3 3-90, 3-98
STMA 9 3-90
STMA 10 3-90f, 3-16, 3-103
STMA 11 3-91
STMA 12 3-91f
STMA 13 3-92f
STMA 14 3-93
STMA 15 3-93f
STMA 16 3-94
STMA 17 3-94
STMA 18 3-94
STMB privileged system calls 10-16f
STMC privileged system calls 10-18ff
STMD privileged system calls 10-20
STOP 3-96, 1-6
STOP, EDIT command 7-18, 7-20
(see stopping an EDIT session)
STOP key (FM template) 5-3f, 5-11
(see also File Maintenance)
stop FM 5-4
(see File Maintenance)
STOP SM command 6-6
stopping an EDIT session 7-18
(see STOP, EDIT command, END, EDIT command, OMIT, EDIT command)
storage, working 1-2ff, 1-5ff, 3-11ff, 3-22, 3-45f, 3-56, 3-78, 3-81f, 3-85, 3-97
string 2-5ff, 1-4, 2-1, 3-20f, 3-25, 3-70f
assign binary number 2-6, 3-7, 3-12, 3-44
assign digits to 2-6, 3-12, 3-44
assignment 2-5ff, 3-25, 3-44
crammed 3-16, 3-90, 3-103, 4-35
delimiter 7-13, 7-15
extract number from 2-6, 3-7, 3-12, 3-83
field formats 5-8
fill with ASCII values 3-25, 3-70f
functions 2-2, 2-6f, 2-8, 3-65
holding binary values 2-6, 3-7, 3-12, 3-44
length of 3-43
literal 2-5ff, 3-44, 3-70f
of digits 2-6, 3-7, 3-12, 3-44, 7-46
uncram 3-103
string/table translations 3-92f
string variable, maximum number of bytes 2-6, 3-20f, 3-25, 3-43
string variables 2-5ff, 2-3, 3-20f, 3-44, 3-73f, 3-104f
dimensioning 3-20f, 3-43
subscripts 2-6f, 3-20f, 3-25, 3-73f
strings
byte positions 2-6f, 3-7, 3-12, 3-20f, 3-25
changing with EDIT utility 7-15
concatenated 4-35
in relational expressions 2-7, 3-32
locating with EDIT utility 7-13
truncate 3-102
\STX DOC command 7-6ff
(see DOC utility)
subdirectories 10-1
creating 8-5, 8-27
subfield, digit (N) 5-8
(see also field formats)
subfile 3-23, 3-48, 3-65, 3-74, 3-104f, 4-4, 4-6f, 4-12f, 4-15ff, 4-18ff, 4-39, 7-35, 7-37
subfile, find, build C1 array 7-22
(see also FINDFILE.SL)
subfile, open 7-35
(see also OPEN utility)
subfile starting location 4-13
(see also byte offset to record 0)
subindex, define/delete (INFOS) 9-2f
subpartitions 4-2
subroutine, assembly language Appendix D
subroutine entry point 2-2, 3-28f, 3-57f, 3-75, 7-1
subroutine files (.SL) 4-2
subroutine line numbers 2-2, 3-28f, 3-57f
subroutine, prewritten 2-2, 3-22, 3-28f, 3-89
subroutine, specific variables 2-2
Subroutines and Utilities Chapter 7
subroutines 2-2, 2-1, 3-22, 3-28f, 3-57f, 3-89, 7-1ff
Subroutines DELREC.SL 3-29, 4-4ff, 4-12ff, 4-30ff, 7-4, 7-25
subroutines, DIV.SL 7-5
subroutines, FINDFILE.SL 4-4, 4-12, 7-22
subroutines, FORM.SL 6-7, 7-23
subroutines, GETCM.SL 7-24
subroutines, GETREC.SL 4-4ff, 4-12ff, 4-19ff, 4-30ff, 7-24
subroutines, INITDEX.SL 4-6f, 4-18ff, 7-32
(see also INDEXBLD, Index Files)
subroutines, MUL.SL 7-34
(see also calculate percent of a number)
subroutines, POSFL.SL 7-37, 3-64f, 4-4, 4-12ff, 4-19ff, 4-30ff, 7-25
subroutines, PROTFORM.SL 7-38, 6-7
subroutines, UNFORM.SL 7-44f, 6-7f
subroutines, VAL.SL 7-46, 2-3, 3-44, 6-7f, 7-23
(see also Screen Handling)

Licensed Material-Property of Data General Corporation

subroutines, merge 7-1ff
subroutines, user written 10-4
(see also USERSUBS Appendix D)
subscripts to array variable 2-4f, 3-20f, 3-73f
subscripts to string variables 2-6f, 3-20f, 3-73f
substring assignment 2-6f, 3-25, 3-44
substring transfers 4-26ff
substrings 2-6, 3-25, 3-73f
substrings, byte positions 2-6f, 3-25
subtraction 2-5
suppress labeling of GOTOs and GOSUBs 3-75, 3-89
suppress zero printing 3-70f
suspend of a LOCK 3-47f, 4-39
SWAP 3-97f, 1-7, 2-1, 3-11, 3-81, 3-86f, 7-3, 8-28, 10-13
swaps to CLI 7-3
switch settings, CPU 3-98
switches, associated with field read 7-24
(see also GETCM.SL subroutine)
symbol table of save file 7-47
SYS 3-98f
SYS(7) - error code 3-98f, 3-24, 3-56f, 3-86f
SYS(10) 3-98, 3-88f
SYS(22) 3-98, 3-101
SYS(26) 3-98, 3-59, 3-88f
SYSDEC (OPCLI command) 10-16
$SYS directory 10-10f
system call decrement 10-16
system call priority 10-16
system calls, privileged 10-16 to 10-21
system calls, RDOS/DOS 10-18
system calls, user 3-86ff, 3-90ff
system channel assignments, display 7-41
(see also SCHANS utility)
system directory 8-18, 8-32
system information 3-98
system library diskette 10-1
System Management Chapter 10
system manager 1-1, 10-1, 10-11
system name, display current 8-18
system operator 1-1, 3-51, 10-1, 10-11, 10-15
system programmer 1-1, 10-11
System Security 10-10ff

T, EDIT command 7-14, 7-20
T (print format) 3-71
TAB expression 3-65ff, 3-68f
TAB TO NEXT (F9), SM function key 6-2f, 6-6
tab 5-4, 7-13
(see also carriage return, File Maintenance, EDIT utility)
tab size 3-67, 3-88, 3-100
TAB statement 3-100, 3-66
TABBUILD utility 5-14
(see also Arrays, quick definition of)
TABLE command 8-37, 8-29
table file (.TB) 4-2, 4-10, 4-18ff, 5-1ff, 5-4ff, 5-12ff
(see also File Maintenance Utility)
table file, file maintenance (FM.TB) 5-2, 5-4ff
table file list, printing 5-12
(see also FMTABPRINT utility)
table file records 5-4ff, 5-14
table file, setting up 5-1ff
table file records, Copy 5-14
table, lock 3-47f, 4-39
(see also File Locking)
table/string translations 3-92f
table of contents, print 7-11
(see also DOCTOC utility)
tabs, setting, with EDIT 7-13
(see also SET, EDIT command)
tabulated print zones 3-65ff, 3-68f, 3-100
tag files 2-8, 4-19ff, 4-25, 7-26, 7-28, 7-42
tape, end of file 3-52ff
tape, magnetic 3-52ff, 3-60f, 8-13, 8-15f, 8-38ff
tape records 3-52ff
tape status 3-52ff, 3-84
.TB extension 5-2
(see also table files)
TBUILD utility 7-42, 4-6f, 4-18ff, 4-25, 7-28
(see also Index Files, tag file)
TCOPY command 8-38
techniques, formatting 2-3
templates, File Maintenance (FM) 5-3
template, SM function keys 6-3
temporary file (.TM) 4-1
TERM utility 7-43, 7-2
(see also change terminal key functions)
terminal bell or alarm 3-68
terminal characteristics 3-67ff, 3-87ff
terminal control expressions (@) 3-34f, 3-36f, 3-65ff, 3-68f
terminal, keys (6053) 3-88
terminal keys, change functions 7-43
(see TERM utility)
terminal line number 7-36
(see also PORTS utility)
terminal keyboard Preface
terminal port number 3-98
terminal, send/receive messages 3-51
terminal type 1-1, 3-86, 10-5, 10-10
terminating value of the loop 3-26f
terminator, program 3-21
terminators, string 3-36f, 3-66ff
test status of deleted records 7-21
(see also FILESORT utility)
text files 2-8, 3-22, 8-29
text files, create 7-11ff
(see also EDIT utility)
text files, edit 7-11ff
(see also EDIT utility)
text files, output by DOC utility 7-6ff, 7-11
text justification, automatic 7-6, 7-8
(see also DOC utility)
text, justifying 7-6ff
(see also DOC utility, \STX command, \ETX command)

Licensed Material-Property of Data General Corporation

TFER command 8-39
THEN 3-32f, 3-11f, 3-56f
timed input 3-101
time of log on 3-98
time remaining after INPUT or TINPUT 3-98
time of day 3-98, 8-19
TINPUT 3-101
TINPUT time remaining 3-98, 3-101
TPRINT command 8-40, 10-16
transfer bytes 3-7, 3-73f, 3-83, 3-104f
transfer data, variables 2-3f, 3-7, 3-73f, 3-104f, 4-26ff, 4-29ff
transfer depending on a condition 3-32f, 3-57f
transfer 512-byte blocks 3-8, 3-10, 4-33f
transfer substrings 4-26ff
transfer to and from subroutines 3-28, 3-57f
translations, ASCII/EBCDIC and EBCDIC/ASCII 3-93
translations, lowercase/uppercase 3-93
translations, string/table 3-92f
trap errors in program 3-24, 3-36, 3-56f
trap interrupts in program 3-59, 3-88ff
TRIPLE (1) PRECISION 10-3
triple precision binary representation 2-3f
triple-precision integer 7-40
(see also number sign in variable name)
triple precision numeric variable 2-3ff, 3-44, 3-73f, 3-104f, 4-26
true/false decisions 3-32f
TRUN$ 3-102, 3-43
truncate number to make integer 3-38, 3-85
truncate strings 3-43, 3-102
$TTO1/TTI1 (second TTY support) 10-1, 10-3
$TTI/$TTO (operating system's console) 10-3
$TTI1/$TTO1 (operating system's console) 10-3
tuning file or report 8-40, 10-1
two dimensional arrays 2-4f, 3-20f
two, powers of 3-6
two's complement 2-3
TYPE command 8-41
type 1 records 5-4ff
(see also File Maintenance)
type 2 records 5-6f
(see also File Maintenance)
type 3 records 5-7
type 4 records 5-6ff
(see also File Maintenance)
type 5 records 5-8f
type 6 records 5-10

UCALL 10-4, Appendix D
(see also user written subroutines, USERSUBS)
UCHANS utility 4-15
(see also Utilities, user channel assignments)
UCHANS utility 7-43, 4-15
UCM$ 3-103, 3-16, 3-90
unary minus sign 2-5
unary plus sign 2-5
unconditional paging 7-7
(see also DOC utility, \PAGE command)
uncramming a string 3-16, 3-103
underscored field 3-68
unedited alphanumeric fields 5-8
UNFORM.SL subroutine 7-44f, 6-7f
(see also Screen Handling)
unformatted fields 6-7
(see also UNFORM.SL)
unformatted screen input/output 7-44
(see also UNFORM.SL)
unique number generated in a series 3-98
UNLINK command 8-41
UNLOCK 3-47f
(see also LOCK)
unmapped NOVA2 10-3
unpend keys 3-67, 3-87f
unprotected fields 6-1, 7-38, 7-44
unsigned integer, conversion 7-46
uppercase ASCII translations 3-93
user 1-1
user attribute 8-6
user channels 4-15, 10-5
(see also UCHANS utility)
user-defined functions 3-18
user directories 4-2
user directory (subpartition) files 4-2
user ID 1-1, 5-7, 10-11
user status table 10-23ff
user status table, address 7-41, 10-23ff
user status word 3-98, 10-23ff
user system calls 3-86ff, 3-90ff
user written subroutines 10-4
(see also USERSUBS Appendix D)
U$ERS file 10-13, 10-11
USERSUBS Appendix D
using File Maintenance (FM)
(see File Maintenance (FM) Using)
using function keys 5-10f
using your screen 6-7ff
utility programs (or utilities) 2-2f, 2-1, 3-11, 3-96f, Chapter 7
utilities, ATTACH 7-2
utilities, CLI BASIC 7-3, 8-1
utilities, DOC 7-6ff, 7-9f, 7-11
utilities DOCTOC 7-11
(see also DOC utility, Print Table of Contents)
utilities, EDIT Preface, 2-8, 3-45, 3-75, 4-28, 7-6, 7-11ff
utilities, error messages Appendix C
utilities, FMLOG 5-1ff
(see also File Maintenance (FM))
utilities, FMPRINT 5-1ff
(see also File Maintenance (FM))
utilities, FMTABPRINT 5-1ff
(see also File Maintenance (FM))
utilities, File Maintenance (FM) 2-2, 4-6, 4-10ff, 4-18ff, Chapter 5
utilities, FILESORT 7-21

Licensed Material-Property of Data General Corporation

utilities, FMLOG 5-1ff, 5-14
(see also File Maintenance)
utilities, FMPRINT 5-1ff, 5-14
(see also File Maintenance)
utilities, FMTABPRINT 5-1ff, 5-12
(see also table file list, printing)
utilities, IBUILD 4-6f, 4-18ff, 4-22ff, 7-26ff
(see also index files)
utilities, INDEXBLD 7-28, 4-6f, 4-18ff, 4-20ff
(see also index files)
utilities, INDEXCALC 4-6f, 7-29
(see also index files)
utilities, INDEXPRT 7-30
(see also print an index)
utilities, INITIFILE 7-31f, 4-4, 4-6f, 4-9f, 4-11ff, 4-19ff, 5-1f
(see also Index Files)
utilities, LOCKS 7-33, 4-39
utilities, LSPEED 7-33
(see also set terminal's speed)
utilities, MOVETABREC 5-14
(see also Table File Records)
utilities, OPEN 7-35, 4-4, 4-9ff, 4-12, 4-15ff
(see also FINDFILE.SL, subfiles, find available channel)
utilities, PORTS 7-36
utilities, QFILESORT 7-39, 4-22
utilities, RELINK 7-40
utilities, RNAM 7-40
utilities, SCHANS 7-41
(see also display system cahnnel assignments)
utilities, Screen Maintenance (SM) 2-2, 5-6, Chapter 6
utilities, TABBUILD 5-14
utilities, TBUILD 7-42, 4-6f, 4-18ff, 4-25
(see also Index Files, tag file)
utilities, TERM 7-43, 7-2
utilities, UCHANS 7-43, 4-15
(see also UCHANS, user channel assignments)
utilities, VAR 7-47
utilities, XBUILD 7-47, 4-6f, 4-18ff, 4-25, 7-28
(see also Index Files)
utility library diskette 10-1
utility source diskette 10-1

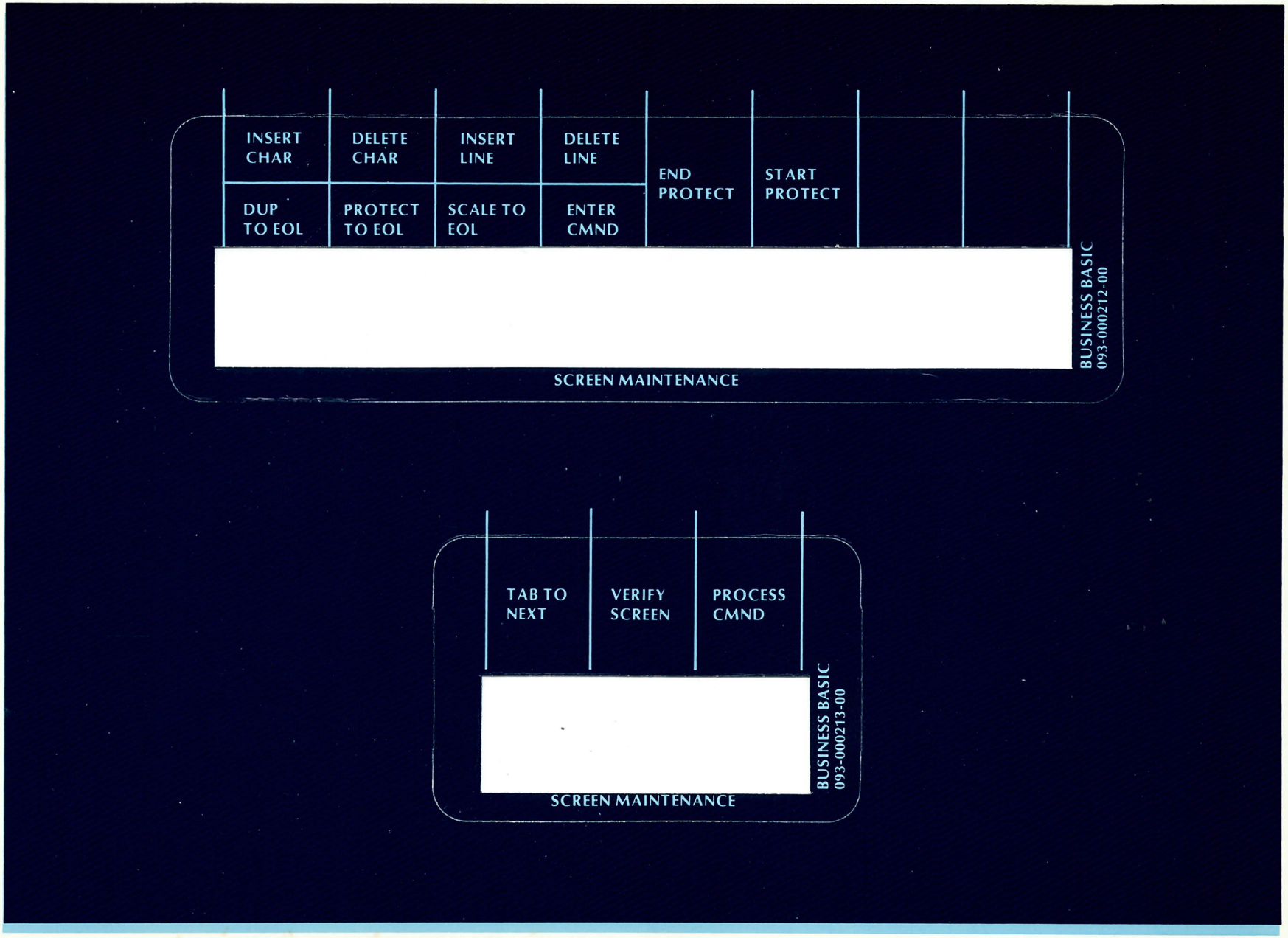
V, EDIT command 7-18, 7-20
(see also verify mode, EDIT utility)
VAL.SL subroutine 7-46, 2-3, 3-44, 6-7, 7-23
(see also Screen Handling)
values, assign to variables 2-3, 3-17, 3-44, 3-72f
values for DATA 3-72f
values for variables 2-3, 3-17, 3-44, 3-72f
VAR utility 7-47
variable, double precision numeric 2-3ff, 3-44, 3-73f, 3-104f
variable names 2-2f, 3-20, 7-40
variable, single precision numeric 2-3ff, 3-44, 3-73f, 3-104f
variables 2-3ff, 1-6, 2-1f, 3-17, 3-44, 3-72f, 3-73f, 3-104f, 7-47
cross reference list 8-29, 8-37
variables, array 2-4f, 2-3, 3-20f, 3-44, 3-73f, 3-104f
variables, assign values 2-3, 3-17, 3-44, 3-72f
variables, correct the sign 4-29ff
Variables, input/output 4-26ff
variables, numeric 2-3ff, 3-44, 3-73f, 3-104f
variables, program, list 7-47
(see also VAR utility)
variables, single precision (%) 4-6, 4-26ff
(see also byte transfers)
variables, specific subroutine 2-2
variables, string 2-3, 2-5ff, 3-20f, 3-44, 3-73f
variables, subscripts for array 2-4f, 3-20f, 3-73f
variables, subscripts for string 2-6f, 3-20f, 3-73f
variables to transfer data 2-3f, 3-7, 3-73f, 3-104f
variable, triple precision numeric 2-3ff, 3-44, 3-73f, 3-104f, 4-26ff
varying number of decimals 7-44
verify mode, EDIT utility 7-18, 7-20
(see also V, EDIT command)
VERIFY SCREEN (F10), SM function key 6-2f, 6-6
VFU command 8-42
virtual overlays 10-6

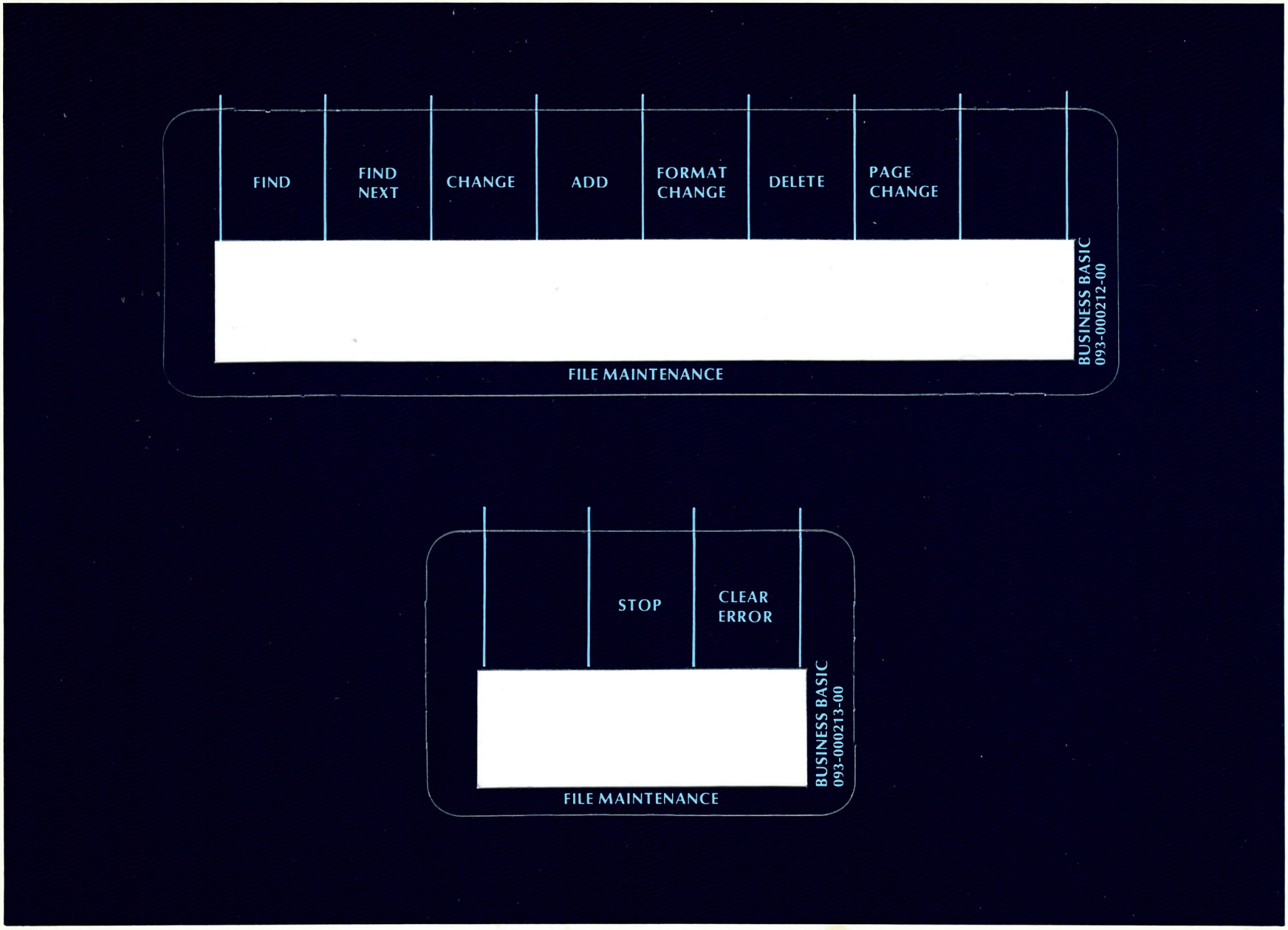
width, page 3-64, 3-100, 8-32
WINDOW MAPPING 10-3
words 2-3ff
working storage 1-2ff, 1-5ff, 3-11f, 3-22, 3-45f, 3-56, 3-78, 3-81f, 3-85, 3-97, 7-3, 7-18
write an INFOS record 9-4
write blocks to file 3-10
write data to file 3-104f
WRITE FILE 3-104f, 2-3, 4-29ff
WRITE, NFOS 9-4
write-protected file attribute 4-3, 8-6
write records to a file 4-29ff, 4-32
(see also WRITE FILE)
WRITE SM command 6-6
write tape records 3-52ff
writing a program 1-3

X (print format) 3-71
XBUILD utility 7-47, 4-6f, 4-18ff, 4-25, 7-28
XCOL SM field definition 6-14, 7-44
XDEC 7-44
(see also FORM.SL)
XFER, BASIC CLI 3-54, 8-45
XFER, RDOS/DOS CLI 3-54
XFLGS SM field definition 6-14, 7-44
XLROW 7-44
(see also screen last line read)
XROW 7-44
XWID SM field definition 6-14, 7-44

Z (print format) 3-71
zero column of array 2-4
zero suppression inprinting 3-70f

Licensed Material-Property of Data General Corporation


INSERT CHAR
DELETE CHAR
INSERT LINE
DELETE LINE
END PROTECT
START PROTECT
DUP TO EOL
PROTECT TO EOL
SCALE TO EOL
ENTER CMND
SCREEN MAINTENANCE
BUSINESS BASIC
093-000212-00
TAB TO NEXT
VERIFY SCREEN
PROCESS CMND
SCREEN MAINTENANCE
BUSINESS BASIC
093-000213-00


FIND
FIND NEXT
CHANGE
ADD
FORMAT CHANGE
DELETE
PAGE CHANGE
BUSINESS BASIC
093-000212-00
FILE MAINTENANCE
STOP
CLEAR ERROR
BUSINESS BASIC
093-000213-00
FILE MAINTENANCE